Ph.D. Thesis

XML-intensive Software Development

Felipe Ibáñez Anfurrutia

2015

eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

onekin

Web Engineering Research Group

Supervisor: Prof. Dr. Oscar Díaz

The author enjoyed a pre-doctoral grant by the University of the Basque Country during the years 1999-2002 and co-financed by the European Social Fund. This research was also supported by both the Secretaría de Estado de Política Científica y Tecnológica of the Spanish Government under contract TIC 1999-1048-C02-02, and the Departamento de Educación, Universidades e Investigación of the Basque Government under contract UE2000-32.

to my wife, Miren,

my daughters, Izaro and Lorea,

eta gurasoei, Felipe eta Bixenta

Summary

The presence of XML is pervasive, yet its youth makes developers face a lot of challenges when using XML in cutting-edge applications. This thesis confronts XML in three different scenarios: document exchange, *Software Product Lines* (SPLs) and *Domain-Specific Languages* (DSLs). Digital document interchange is one of the prominent applications of XML. However, business documents frequently hold derived data, i.e. data which is calculated from other data. Here, we face the question of how can XML Schema be extended to account for derived data. This dissertation proposes the *XDerive* namespace that permits deriving functions to be integrated as part of XML Schema documents. Next, SPLs offer an approach to develop a family of software products by reuse. What if these products are realised as XML documents? The thesis addresses how *Feature-Oriented Programming* (an approach to SPL development) can be extended to account for XML specifics (e.g. tag-based description, validation awareness). Finally, XML in Web development. The increasing growth in size and complexity of web sites calls for a systematic way to web sites development. *Leaflet websites* are a kind of content-oriented websites. *Domain Specific Languages* (DSL) are usually geared towards a specific domain or application, offering only a restricted suite of notations and abstractions. Here, we address the domain of *leaflet websites,* i.e. websites meant for static-content navigation, and introduce an XML-based DSL: *XLeaflet*. A distinctive feature of *XLeaflet* is its architecture: thick-browser. This can account for an important reduction in the network traffic. These so-different fields (i.e. document exchange, SPLs and DSLs) act as "stress tests" to assess the ductility of XML concepts and technology to cope with so heterogeneous environments.

Resumen

La presencia de XML es un fenómeno generalizado. Sin embargo, su juventud hace que los desarrolladores se enfrentan a muchos desafíos al utilizar XML en aplicaciones de vanguardia. Esta tesis enfrenta XML a tres escenarios diferentes: intercambio de documentos, *Líneas de Producto Software* (LPS) y *Lenguajes eSpecíficos de Dominio* (LSD). El intercambio digital de documentos es una de las aplicaciones mas importantes de XML. Sin embargo, los documentos de negocios con frecuencia tienen datos derivados, es decir, datos que se calculan a partir de otros datos. Aquí, nos enfrentamos a la cuestión de cómo ampliar el XML Schema para capturar datos derivados. Esta tesis propone un nuevo vocabulario: *XDerive*. *XDerive* permite expresar funciones derivadas teniendo en cuenta las especificidades de XML. Por otro lado, las LPS ofrecen un enfoque para desarrollar una familia de productos de software mediante la reutilización. ¿Qué pasa si estos productos se realizan como documentos XML? La tesis aborda cómo ampliar la programación orientada a características (un enfoque para el desarrollo de LPS). También considera XML en el desarrollo Web. En concreto, abordamos el desarrollo de aplicaciones web utilizando lenguajes específicos de dominio. En nuestro caso, el dominio es de los sitios web orientados a la navegación de contenidos estático. Con este fin, diseñamos el lenguaje XLeaflet. Una característica distintiva de XLeaflet es su arquitectura: “thick-browser”. Objetivo: conseguir una reducción importante en el tráfico de red. Estas tres áreas (es decir, de intercambio de documentos, LPSs y LSDs) sirven como "pruebas de resistencia" para evaluar “la ductilidad” de los conceptos y la tecnología XML para hacer frente a entornos tan heterogéneos.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

XML, the short for eXtensible Markup Language, started life in February 1998. It was brought into existence partly as an answer to the growing limitations of HTML and also to introduce an open standard of web interoperability and data sharing. The aim: to separate the structure of data from its format. While HTML is intended for content presentation but not the type or structure of said information, XML is geared entirely towards the structure and not the presentation of data [BPSM98].

XML is a meta-markup language, i.e. it can be used, fundamentally, to create mark-up languages. The power of this feature is phenomenal and provides almost limitless applications of the language. Indeed, XML artefacts play a preponderant role in current software practises, specially in the Web setting. XML can be found in both code artefacts (e.g. rendering markup languages, ANT configuration files[Thea], Struts controllers[Theb], etc), and non-code artefacts (e.g. deployment descriptors such as *portlet.xml*[JCP03], UML diagrams serialised through XMI[Obj], asset documentation through RAS[Obj05]).

The presence of XML is pervasive yet its youth makes developers to face a lot of challenges when using XML in cutting-edge applications. Many XML technologies are still in the developing stage, and it will take some time until there is a consistent, time-tested, universally accepted framework for XML-centric application design and development. Until then, application designers and developers are mostly on their own when it comes to choosing the most appropriate technologies for designing and implementing XML-oriented information systems.

This dissertation promotes a document-centric approach to both development and delivery of the application logic in a WWW setting. To this end, new *Domain-Specific Languages (DSL)* are defined as well as extension mechanisms to existing tools (e.g. parser, browsers) in order to support them to account for new scenarios. In this sense, this dissertation can be described as "scenario driven" in the sense that the research questions addressed rise within a certain practice. A practice is "a set of human activities performed regularly and seen as meaningfully related to each other by the people participating in them" [JP14]. From the use of XML in non-conventional practices emerge limitations to the current tooling and standards on XML. How to overcome these limitations has been the endeavour of the author during these last years. His findings, proposals and proof-of-concept applications are the main content of this dissertation.

The three scenarios or practices being considered are: XML for document exchange, XML in Software Product Lines, and XML in Web development.

1.2 XML for document exchange

Document interchange is one of the prominent applications of XML. A business document is *"a unit of business information exchanged in a business transaction"* [Eur04]. Order, billing or delivery forms are all examples of business documents that collect data which give support to a certain business function, regardless of how this data is finally stored. Standards emerge to normalize this description: UBL[ubl04], EDIFACT[edi04], X12[x1204], IDA[Eur04], to name a few. For instance, IDA is a European program which aims at specifying business documents exchanged between a public sector buyer and an external supplier during the ordering and invoicing phases [Eur04]. XML is becoming the standard for document description in B2B settings. And *XML Schema* is gaining wide acceptance as the schema language to define the structure of these documents. Indeed, IDA uses *XML Schema*.

Business documents frequently hold derived data, i.e. data which is calculated from other data. For example, the *totalAmount* of an order can be obtained from the cost of each item included in the order minus some applicable discount. Such *deriving functions* support important business policies, including terms and conditions (e.g. policies for price discounting), service provisions (e.g. policies for refunds), or surrounding services (e.g. policies for lead time to place an order). Such business policies are the subjects of contracts among partners, and they can normally be found in catalogs, storefronts and marketplaces, as well as in bids and requests communicated during negotiations, procurements and auctions.

The research question is:

how can XML Schema be extended to account for derived data?

1.3 XML in Software Product Lines

So far, most Web applications are conceived in a one-to-one basis. A recent study indicates a cloning rate (i.e. code repetition throughout the application) of 17-63% within Web applications of the same organisation [RJ05]. This cloning evidences the existence of a common, although implicit, theme throughout the applications, and confirms an intuition felt in most organisations: code similarities among applications. These similarities are being handled in various ways such as IFDEFs, configuration files, installation scripts or cloned software copies *à la "copy-paste-modify"*. However, these solutions do not scale well and can hinder maintenance as the number of variations increases.

Software Product Line (SPL) techniques strive to make this recurrent functionality explicit so that reuse is facilitated without compromising maintenance. Hence, it may be appropriate to support Web applications as outputs of an SPL. Rather than constructing every product from scratch, a product line is first constructed, and then, the products are obtained after the product line. By using SPLs to manage variations in their code, organisations are reporting order of magnitude reductions in time-to-market, engineering overhead, error rates and cost[1].

One technique to handle variations is step-wise refinement [BSR04]. Step-wise refinement is a powerful paradigm for developing a complex program from a simple program by incrementally adding details. This approach attempts to depart from current “clone&own” practise by leveraging reuse of the common parts, and separating variable and changing parts as *program deltas*. The final product is obtained through composition: the common parts are leveraged with the program deltas that realise the variations for the product at hand.

This work addresses the use of deltas (i.e.refinements) as a modularisation technique for XML artefacts. It is worth noticing that the nature of refinement depends on the artefact being refined, e.g. Java artefacts are not necessarily refined in the same way that XHTML artefacts. When the artefact is *“.java”*, a class refinement can introduce new data members, methods and constructors to a target class, as well as extend or override existing methods and constructors of that class [BSR04]. But, what is meant to refine an XML artefact? Does it mean that we can arbitrarily insert or delete a node anywhere in an XML document tree?

The research question is:

how to introduce deltas into XML artefacts?
how are XML deltas defined, composed and validated?

1.4 XML in Web development

Web application development is currently suffering from a severe bottleneck as the gap between available implementation tools and application’s requirements is enlarging. But these difficulties are likely to become even more stringent as web masters have to face maintenance. This is felt specially urgent in the area of e-commerce. In today´s e-commerce world, companies should adapt to changing conditions and rapid evolution. However, it is a frustrating experience to see how often the web site bottleneck slows and restricts the evolution of the organization the site is supposedly serving.

[1]Refer to *www.softwareproductlines.com/benefits/benefits.html* for a detailed account.

In response to the previously described need, distinct projects have been launched which aim at providing design guidelines and supporting tools for systematic web site construction. One of the most frequently cited guideline is to split requirements into content concerns, navigation concerns and presentation concerns by using a model-based approach [PM00].This approach aims to find declarative models, preferably orthogonal, that allow designers to declaratively specify a specific concern of the application without being immediately immersed in details of implementations. A design is then conformed by a set of schemata (i.e. model instances) which describe distinct aspects of the application. Declarativeness and orthogonality accounts for maintainability: the separation of concerns and their declarative specification allow to easily update a schema while minimizing the impact on the rest of the application. DSLs are usually geared towards a specific domain or application, offering only a restricted suite of notations and abstractions. Hence, this DSL vision will not be possible in a general basis but it could happen for specific domains. Specifically, we look at static, content-oriented websites, hereafter referred to as ***"leaflet websites"***. Conference websites, product catalogues or websites with content about a teaching course are the kind of websites we are tackling. Here, the challenge rests more on rendering and navigation rather than supporting transactional-like functionality (e.g. purchases, enrollments and the like).

The research question is

> *could "leaflet websites" be developed with the only help of a DSL?*

1.5 Research Methodology

Design science is "the scientific study and creation of artefacts as they are developed and used by people with the goal of solving practical problems of general interest" [JP14]. In additional quote from P. Johannesson and E. Perjons, brilliantly summarise the essence of this approach:

> *The starting point for a design researcher is that something is not quite right with the world, and it has to be changed. A new arte-*

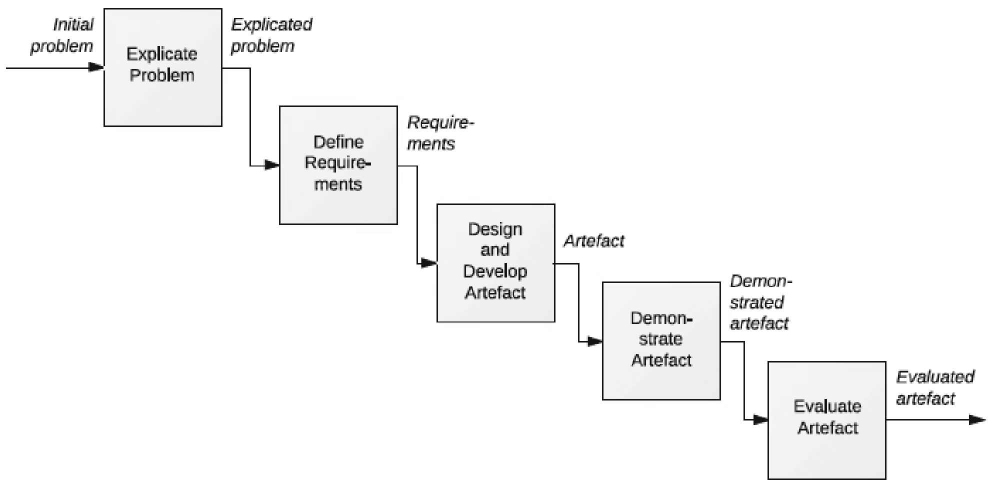


Figure 1.1: Overview of the method framework for design science (taken from [JP14])

> *fact should be introduced into the world to make it different, to make it better. Design science researchers do not only think and theorise about the existing world. They model, make, and build in order to create new worlds. They produce both a novel artefact and knowledge about it and its effects on the environment. In particular, they need to formulate problem statements, determine stakeholder goals and requirements, and evaluate proposed artefacts. In other words, artefacts as well as knowledge about them are research outcomes for design science.*

Specifically, we follow the framework defined in [JP14]. For self-containtion, next paragraphs are an excerpt of [JP14] where the different tasks of their Design Science methodology are described (see Figure 1.1):

- **Explicate Problem.** The Explicate Problem activity is about investigating and analysing a practical problem. The problem needs to be precisely formulated and justified by showing that it is significant for some practice. The problem should be of general interest, i.e. significant not only for one local practice but also for some global practice. Furthermore, underlying causes to the problem may be identified and analysed.

- **Define Requirements.** The Define Requirements activity outlines a solution to the explicated problem in the form of an artefact and elicits requirements, which can be seen as a transformation of the problem into demands on the proposed artefact.

- **Design and Develop Artefact.** The Design and Develop Artefact activity creates an artefact that addresses the explicated problem and fulfils the defined requirements. Designing an artefact includes determining its functionality as well as its structure.

- **Demonstrate Artefact.** The Demonstrate Artefact activity uses the developed artefact in an illustrative or real-life case, sometimes called a "proof of concept", thereby proving the feasibility of the artefact. The demonstration will show that the artefact actually can solve an instance of the problem.

- **Evaluate Artefact.** The Evaluate Artefact activity determines how well the artefact fulfils the requirements and to what extent it can solve, or alleviate, the practical problem that motivated the research.

As indicated by P. Johannesson and E. Perjons, these tasks do not follow strictly in sequence. Rather, research is commonly iterative, moving back and forth between all the activities of problem explication, requirements definition, development, and evaluation. The arrows in Figure 1.1 should not be interpreted as temporal orderings but as input–output relationships. In other words, the activities should not be seen as temporally ordered but instead as logically related through input–output relationships.

1.6 Outline

This thesis has been developed in the context of the *AtariX* and *Kimu-Berri* projects, collaborations between the University of the Basque Country, LKS S. Coop. and Mondragon Sistemas de Información(MSI) companies. This pushes us to achieve not only an academic contribution but also to look at the applicability of these ideas in an industrial setting. This introduced an important risk through the thesis, the need of implementing the ideas. In this way, this thesis follows an extensive exploration of the issue at hand (i.e. XML) rather than a deep, drill-down approach of a specific concern. This thesis is composed of five chapters, including this one, that each target specific research questions:

1. *how can XML Schema be extended to account for derived data?* Chapter 2
2. *how to introduce deltas into XML artefacts? how are XML deltas defined, composed and validated?* Chapter 3.
3. *could "leaflet websites" be developed with the only help of a DSL?* Chapter 4

The chapters can be read independently. Specifically,

Chapter 2 discusses that the calculation of derived data in business documents is hidden to the partner of e-commerce and proposes an approach to externalize the semantics of deriving functions. To this end, an XML-Schema and JAXP-compliant XML parser are extended.

Chapter 3 elaborates on the notion of XML delta in order to realize feature variability in a SPL setting. A vocabulary is introduced for defining XML deltas which can then be validated and composed to output enhanced XML documents. Composition synthesises a new document from a base document and an XML delta. AHEAD is selected to deal with such composition. A challenge is whether a validity of an XML delta with respect to an schema S guarantees that synthesised documents are valid with respect to S, provided the refined document is also S compliant. Two use cases are also presented in the following sections:

Chapter 4 presents *XLeaflet*, a model-based tool for content-oriented websites development that renders HTML pages from the declarative schemata specified by the designer. Each concern (i.e. content, navigation, presentation and adaptation) is separately described in an XML document. A distinctive feature of *XLeaflet* interpreter is its architecture, i.e. thick-browser.

Chapter 5 concludes the dissertation. It summarizes the main contributions and publications as well as suggests the opportunities for future research.

Chapter 2

XML for document exchange

2.1 Overview

Business documents such as those currently found in B2B applications, frequently comprise derived elements, i.e. elements whose content can be calculated through a deriving function that examines the content of other elements. Despite its wide presence, the notion of derived elements is not yet supported in XML Schema. This is the challenge addressed in this chapter: how to leverage XML Schema with derived data. To this end, we introduce an artefact: the *XDerive* namespace that permits deriving functions to be integrated as part of XML Schema documents.

The chapter is organized along common Design Science activities: explicate problem, define requirements, artefact design, artefact implementation and artefact demonstration. Related work and conclusions end the chapter.

2.2 Problem Explanation

In which practice does the problem appear? Business-to-Business (B2B) applications (e.g. ERP) are main Information Systems. Here, document interchange

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           elementFormDefault="qualified">
  <xs:element name="order" type="orderType"/>
  <xs:complexType name="orderType">
    <xs:sequence>
      <xs:element name="orderDate" type="xs:date"/>
      <xs:element name="orderNumber" type="xs:short"/>
      <xs:element name="customer" type="customerType"/>
      <xs:element name="lineItem" type="lineItemType"
                  maxOccurs="unbounded"/>
      <xs:element name="shippingData" type="shippingDataType"
                  minOccurs="0"/>
      <xs:element name="subTotal" type="xs:decimal" minOccurs="0"/>
      <xs:element name="applicableDiscount" type="xs:int" minOccurs="0"/>
      <xs:element name="vat" type="xs:decimal" minOccurs="0"/>
      <xs:element name="total" type="xs:decimal" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="shippingDataType">
    <xs:sequence>
      <xs:element name="shippingAddress" type="addressType"/>
      <xs:element name="shippingSpeed" type="speedType"/>
      <xs:element name="shippingPreference" type="preferenceType"/>
      <xs:element name="shipments" type="xs:int" minOccurs="0"/>
      <xs:element name="insuranceCost" type="xs:float" minOccurs="0"/>
      <xs:element name="shippingCost" type="xs:float" minOccurs="0"/>
      <xs:element name="deliveryTime" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <!–content omitted –>
</xs:schema>
```

Figure 2.1: An XML Schema for *order* documents.

plays a preponderant role. For the purpose of this dissertation, the term **"business document"** denotes *"a unit of business information exchanged in a business transaction"* [Eur04]. Order, billing or delivery forms are all examples of business documents that collect data which give support to a certain business function, regardless of how this data is finally stored. Many projects or standards (e.g. UBL[ubl04], EDIFACT[edi04], X12[x1204], IDA[Eur04]) have developed electronic order and invoice messages. For instance, IDA is an European initiative which aims at specifying business documents exchanged between a public sector buyer and an external supplier during the ordering and invoicing phases [Eur04].

Business documents frequently hold **derived data**, i.e. data which is calculated from other data. For example, the *totalAmount* of an order can be obtained from the cost of each item included in the order minus some applicable discount. Such **deriving functions** support important **business policies**, including terms and conditions (e.g. policies for price discounting), service provisions (e.g. policies for refunds), or surrounding services (e.g. policies for lead time to place an order). Such business policies are the subjects of contracts among partners, and can normally be found in catalogs, storefronts and marketplaces, as well as in bids and requests communicated during negotiations, procurements and auctions. XML is becoming the standard for document description and exchange in B2B settings, and *XML Schema* is gaining wide acceptance as the schema language to define both the structure and constraints of these documents. Indeed, the IDA initiative uses *XML Schema*.

What is the problem and the negative consequences of not addressing it? The research question is:

how can XML Schema be extended to account for derived data?

Despite its wide presence, XML Schema does not have yet a mechanism to describe derived elements. This leads to deriving functions being hard-coded into the applications which process the document. For example, deriving functions can be hidden within XSLT templates, or as scripts within the application which process the XML document. But hard coding these functions not only hinders development, it also jeopardizes maintenance. Hard coding, distributing and repeating these functions along the applications that process XML documents would certainly complicate the modification of deriving functions, and thus, of the business strategies these functions support.

In addition, leveraging XML Schema to account for deriving functions is akin to the strategy of externalization promoted by XML Schema. That is, while XML documents let you externalize data representations in a platform independent manner, *XML Schema* let you externalize the types and structural constraints that govern XML documents. Likewise, moving the deriving functions to the *XML Schema* (i.e. externalizing derivation), saves the trouble of implementing (and maintaining) these constraints in each application.

Next section outlines a solution to the explicated problem in the form of an artefact and elicits requirements, which can be seen as a mutation of the problem into demands on the proposed artefact.

2.3 Requirement Definition

Our proposal to address the above mentioned problem is providing new **constructs** to leverage XML Schema with deriving function definition. The B2B setting in which this challenge is framed, raises the following environmental requirements: expressiveness, understandability, interoperability and completeness. Next subsections describe each of these requirements.

2.3.1 Expressiveness

Expressiveness refers to the degree to which a set of constructs is capable of representing the entities of interest in a domain. This subsection introduces the expressiveness challenges with the help of an example.

An *order* document is taken as an example. An order contains information about the customer who placed the order, the order number and the individual line items on the order including their quantities and product references. Figure 2.2 gives an example using an XML vocabulary defined by the XML Schema in Figure 2.1.

Business documents such as the previous one, commonly comprise derived elements (shown in bold in Figure 2.2) whose values are computed according to business policies. Next, four examples are shown, which have been taken or inspired by policies stated on the *Amazon's* site[1]. As we go on, we will raise several issues such as the need for keeping track of the deriving elements being used, and the need for prioritized conflict handling.

[1]http://www.amazon.com

```xml
<?xderive-derivationTrack href="order.derivationTrack.xml" type="text/xml"?>
<order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="order.xsd">
  <date>2002-02-22</date>
  <orderNumber>22222</orderNumber>
  <customer>
    <id>123</id>
    <customerData>
      <name>Tom Hanks</name>
      <billingAddress>
        <address>Oxford Street, 100</address>
        <city>London</city>
        <zip>5555</zip>
      </billingAddress>
    </customerData>
  </customer>
  <lineItem>
    <ref>TV-1</ref>
    <quantity>1</quantity>
    <partialCost>1000.00</partialCost>
  </lineItem>
  <lineItem>
    <ref>Hi-Fi-1</ref>
    <quantity>1</quantity>
    <partialCost>600.00</partialCost>
  </lineItem>
  <lineItem>
    <ref>PC-1</ref>
    <quantity>1</quantity>
    <partialCost>1200.00</partialCost>
  </lineItem>
  <shippingData>
    <shippingAddress>
      <address>Eliseus Camps, 50</address>
      <city>Paris</city>
      <zip>22222</zip>
    </shippingAddress>
    <shippingSpeed>One Day Shipping</shippingSpeed>
    <shippingPreference>few shipments as possible</shippingPreference>
    <shipments>2</shipments>
    <insuranceCost>7.99</insuranceCost>
    <shippingCost>23.97</shippingCost>
    <deliveryTime>Your order will arrive on 2002-2-26</deliveryTime>
  </shippingData>
  <subTotal>2800.00</subTotal>
  <applicableDiscount>20</applicableDiscount>
  <vat>448.00</vat>
  <total>2711.97</total>
</order>
```

Figure 2.2: An *order* document that includes derived elements (in bold).

Example 1. Derived element: `<subTotal>`

- *(Rule) The subTotal is found adding all the partialCosts of the ordered items*

This is an example where the deriving elements (i.e. `<partialCost>`) are local to the document, and only one rule is needed to work out the derived value (i.e. `<subTotal>`). Notice also that derived elements can be based on other derived elements (e.g. `<partialCost>` is in turn a derived element).

Example 2. Derived element: `<partialCost>`

- *(Rule) The partialCost of each item is calculated by multiplying each ordered quantity by the corresponding unitPrice*

Strictly speaking, a derived element should not provide any new information except that derived from other elements in the document. However, such an approach is too restrictive to reflect most of the policies that regulate current business transactions. Most policies consult a business state that is not always reflected in the document itself but in other data resources (e.g. databases, other XML documents such as catalogs, etc). Hence, we extend the potential deriving elements to any data resource available to the document. In this example, the *unitPrice* is not defined in the document itself but retrieved from the company's database.

This externality prevents the document receiver from having a complete picture of how the derived elements have been calculated, and hence, makes the verification of the fulfilment of the agreement (reflected through the deriving function) difficult. Besides, such external elements evolve independently of the document itself so as to impede the re-creation of the business state at the time the document was generated (e.g. at the time the order took place).

This situation is tackled through the *derivationTrack* document. Such a document records a snapshot of the external deriving elements at the time the derived elements were calculated. For the `<partialCost>` example, this implies that the *unitPrice* of the items is recorded in the *documentTrack*. The receiver of the order can then consult this attachment to verify the correct application of the business policies. Section 2.5.1.1 addresses this issue.

Example 3. Derived element: `<deliveryTime>`

- *(Rule A) If the shipping speed is "Standard Shipping" then, the delivery time will be between 4 and 8 days*
- *(Rule B) If the shipping speed is "Two days Shipping", and at least three items are available within 24 hours, then the delivery time will be 3 days*
- *(Rule C) If the shipping speed is "One day Shipping" and at least three items are available within 24 hours, then the delivery time will be 2 days*
- *(Rule D) If the shipping speed is either "One day Shipping" or "Two days Shipping", then the delivery time will be 4 days*
- *(Priority Rule) If both Rules B and D apply, then rule B will "win", i.e. the delivery time will be 3 days*
- *(Priority Rule) If both Rules C and D apply, then rule C will "win", i.e. the delivery time will be 2 days*

Here, the delivery policy is realized as a set of rules. We say a rule is "applicable to fire" when the rule's condition (i.e. the antecedent) is satisfied. When more than one rule can be applicable to fire, a set of (conflicting) values are generated. This implies that the system must resolve the inconsistency and decide on a unique value that is assigned to an element or attribute. For this task, a prioritization schema must be provided. In this case, the literature favours two approaches, namely, *value-based* and *rule-based* [IS92, Hea99]. The former provides a combining function to "merge" all or "select" one of the returned values by each applicable rule. This means that all applicable rules are executed. By contrast, a *rule-based* prioritization schema selects only one rule to execute (e.g. realized by assigning a priority to each rule, and then, the rule with the highest priority "wins"). However, this schema requires a global prioritization schema that is difficult to maintain for large or evolvable rule sets. Therefore, in a B2B setting we favour a partial prioritization schema where conflicts are resolved among smaller sets. This is the role played by the *Priority Rule* in this example.

Example 4. Derived element: `<insuranceCost>`

- *(Rule A) If any item exists whose category is "cd", then the insurance cost will be 1.99 euros*
- *(Rule B) If any item exists whose category is "book", then the insurance cost will be 2.99 euros*
- *(Rule C) If any item exists whose category is "electronic", then the insurance cost will be 5.99 euros*
- *(Rule D) If any item exists whose category is "electronic" and needs special hanging, then the insurance cost will be 6.99 euros*
- *(Priority Rule) If more than one rule apply, the final insurance cost will be the maximum of the returned values*

These rules realize the insurance policy. An order can contain distinct categories of items. Each category has a distinct insurance cost. Hence, several rules can apply simultaneously (e.g. if the order contains "cd" and "book", then "*Rule A*" and "*Rule B*" will be applied). Unlike the previous example, now a *value-based* prioritization schema is followed. That is, the final insurance cost is not calculated as the result of a single rule but as the combination of the returned values by all applicable rules as stated by the *Priority Rule*.

Summing it up, deriving data can be local or remote to the document itself, while the prioritization schema can be *value-based* or *rule-based*. The latter is expressed in the form of meta rules. Such meta-rules also convey important business policies. From an engineering viewpoint, a prioritized schema enables significantly easier modification and more modular revision, both key features to face in the evolvable nature of the B2B setting. Indeed, it has been reported that business policies are among the most evolvable software artifacts found in current information systems. New rules (i.e. policies) can be added, or meta-rules (i.e. strategies) can be updated with no impact to the rest of the rules. Such aspects enhance modularity and facilitate "locality in revision" [GLC99].

Figure 2.3 summarizes the above expressiveness requirements using a feature diagram notation [Kea90]. The feature diagram describes the domain concepts

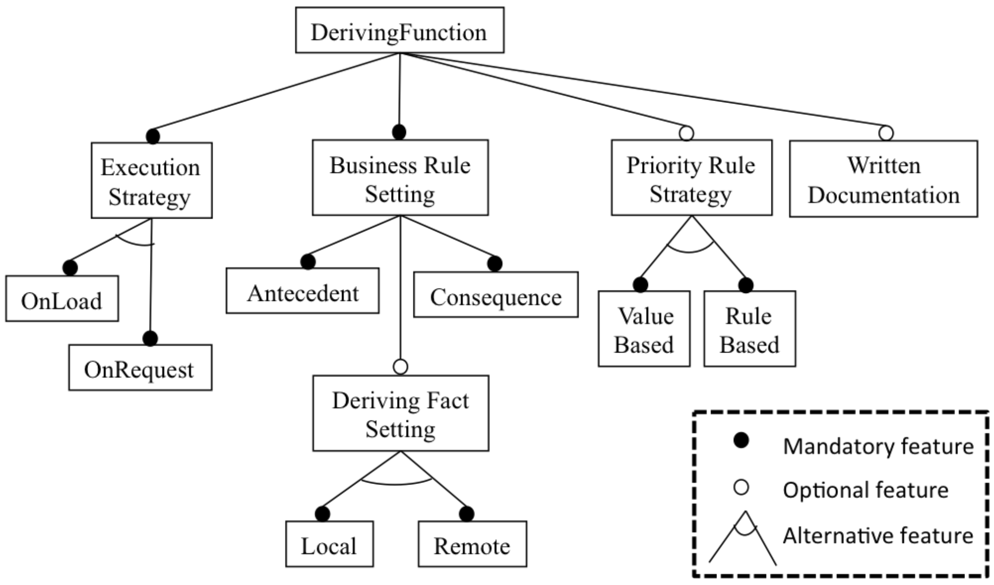


Figure 2.3: Feature-diagram for deriving function domain.

and their interdependencies. In a *Domain Specific Language* (DSL) context, a feature diagram serves to state the commonalities and variabilities of the domain at hand, so that commonalities are built-in into the DSL engine whereas variabilities are supported as parameters to be set by the DSL user [MHS05]. For our case, main features are:

- *Execution Strategy*. This feature deals with the problem of determining when to calculate the derived data: *onLoad*, when an XML document is parsed and loaded in memory; or *onRequest*, i.e. at the moment that it is used and requested to the parser.

- *Business Rule Setting*. A rule includes an antecedent or condition describing the situation where the rule applies, and a consequence, which indicates how to calculate the value of the derived element if the antecedent is met. A rule can be referenced both local or remote deriving facts. In case of remote deriving facts, the source (e.g. DDBB query) of these deriving facts must be provided.

- *Priority Rule Strategy*. When more than one rule can be applicable, two options would be available: "value-based" and "rule-based".

- *Written documentation*. This feature consider the need of describing the deriving function in a human-readable way.

2.3.2 Understandability

We are in a B2B setting: document interchange between different partners. Enhancing partner trustworthiness and hindering repudiation calls making explicit important business rules that govern the interaction between business partners. The so-called *contract* vocabularies focus on capturing the discovery-negotiation-execution life-cycle that models a business transaction [GLC99]. These agreements are frequently reflected in terms of business policies, of which deriving functions are a special realization. Therefore, deriving functions as containers of business policies need to be declaratively described to easy understanding.

2.3.3 Interoperability

This is a requirement coming also from the B2B setting. The solution should not hinder the exchange of document in terms of the infrastructure needed to process those documents. Specifically, document validation should not be compromise by the fact of a document containing deriving functions. This is a kind of backward compatibility in the sense that partners that are *not* concerned with deriving functions can still interact with other partners that use that functions. This rules out solutions based on `<xs:redefine>` like the one presented in [MCV04]. Extending *XML Schema* through `<xs:redefine>` accounts for a cleaner solution but, unfortunately, can refrain the schema from being shared with other partners unless their parsers have been similarly upgraded to account for the same *redefine* statements. This forces partners to keep their XML installation in sync. Therefore, supporting deriving functions should be achieved in a way that is backward compatible, i.e. not forcing all partners to "upgrade" to deriving functions.

2.3.4 Completeness

This requirement is set by IDA itself in the following terms: *"for clarity purposes, all business documents should as far as possible contain all information (reitera-*

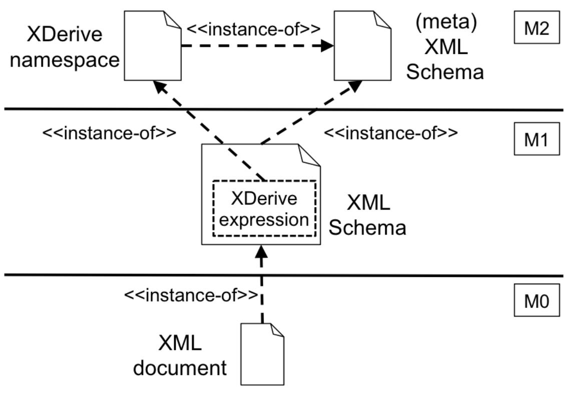


Figure 2.4: *XDerive* design: the annotation approach.

tion of items ordered, parties involved, etc). We thereby ensure that each business document is self-sufficient in interpreting it" ([Eur04], page 15th). Deriving functions consult *deriving data*. Deriving data is not limited to facts explicitly stated on the document, but also include external data to the document itself. For instance, *applicableDiscount* can be obtained from distinct business policies, such as *"5 percent discount if buyer has a history of loyal spending at the store"* or *"20 percent discount if the product is on offer"*. In this example, *applicableDiscount* has two deriving facts (i.e. the consumer's loyalty history, and the product state), that are kept outside the *order* document. This makes business partners other than the issuer, have a partial picture of how the *applicableDiscount* has been obtained, since the information about the state of the product (i.e. whether it is on offer or not) is not recorded in the order. Such a situation might hamper trust construction between partners. Besides, in case of conflicts, it will be difficult to resolve about which discount was applied, as the business document misses important facts.

Next section addresses the design of XDerive, an artefact that addresses the explicated problem and fulfils the previously defined requirements.

2.4 Artefact Design: *XDerive* namespace

This section introduces *XDerive*, an XML Schema namespace for derived element description, and how the *XDerive* expressions should be used. This section tackles the expressiveness requirements set in subsection 2.3.1.

Using XML for schemas and instances instead of using other data formats is beneficial with respect to *interoperability*, *openness*, and *integration*. This means that schemas and instances described with XML syntax are accessible under various platforms and environments, they can be extended by employing XML namespaces, and they can be integrated with other XML standards such as XLink, XSLT, and RDF.

The semantic expressiveness of XML Schema, the schema language recommended by the W3C, however, is not sufficient to define the semantics of derived data. Figure 2.4 shows how we have extended XML Schema in order to enrich XML with derived data. In one hand, the XDerive namespace is described as a (meta)schema (i.e. concepts and constraints) for deriving functions by means of an XML Schema. In the other hand, the XDerive expressions are embedded in an XML Schema, where the business vocabulary (i.e. data type) is described. Thus, an XML Schema can be validated using a conventional XML Schema validator, assuring interoperability with a partner's legacy applications. Moreover, the same XDerive expressions are used for all document instances of the schema.

2.4.1 How to describe deriving functions?

A namespace, *XDerive*, is defined to specify all the aspects addressed in Figure 2.3. Figure 2.5 shows the XML elements and the types that define the *XDerive* namespace. The full XML schema definition can be found in appendix A. Thus, it is possible to reuse in any other XML Schema. Next paragraph delves into the details of the main XML elements:

- *derivingFunction*. This element describes the deriving function. Its content includes a description of its purpose (i.e. *documentation*), a set of *rule*s (1..n) that realize it, and the *prioritizationSchema*, in case that more than one rule is defined. Moreover, the *actuate* attribute holds the execution strategy options: *onLoad (*by default*)* or *onRequest*.

- *rule*. A rule specification includes a description of its purpose (i.e. *documen tation*), the rule's antecedent (i.e. the *test* attribute) and the rule's consequence (i.e. *action* element). The *test* attribute holds an XPath expres-

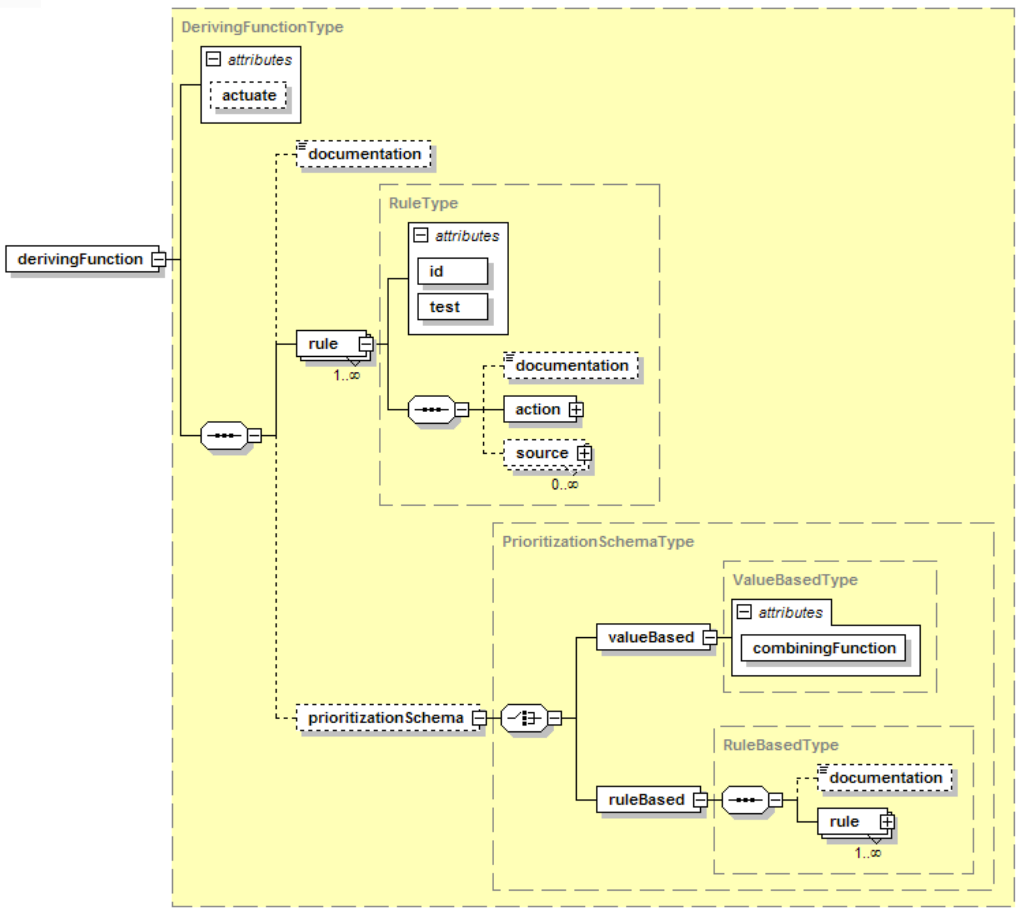


Figure 2.5: *XDerive* namespace: a derivingFunction element provides a content model as *documentation*, *rules* and *prioritizationSchema*.

sion, i.e. a predicate on the content of the instance document. The `action` element comprises a set of standard XSLT instructions (e.g. `<xsl:value-of select="...">)`. This element has a constraint that the latest instruction **must be** `<xsl:value-of ...>` for simple type values and `<xsl:copy-of ...>` for complex type values. In case that remote deriving facts are used in the rule description, more than one `source` element should be defined, one for each distinct remote fact.

- `prioritizationSchema`. This element holds one of the alternatives of a conflict resolution strategy: `valueBased` or `ruleBased` . The former defines a function (i.e. `combiningFunction` attribute) that combines all the result of activated and executed rules. The latter is implemented as a meta-rule by defining a set of *rules* in order to select one of the activated rules.

2.4.2 How to embed *XDerive* expressions?

When describing derived elements, the first issue is the cardinality of derived elements. *XML Schema* provides the `minOccurs` and `maxOccurs` attributes to indicate the allowable occurrence interval of a given element. However, it is not clear what this interval should be for derived elements. Consider the `<insuranceCost>` element, if `minOccurs` is set to 1 the user is forced to introduce the `<insuranceCost>`. But this is not the expected behaviour. On the other hand, if `minOccurs` is set to 0 the `<insuranceCost>` becomes optional, which is also not the right interpretation, particularly for the processing application.

As a result, the schema validator of a "derivation aware" parser should reinterpret the meaning of `minOccurs` when a deriving function has been defined inside the element's definition. In our implementation, `minOccurs` of derived elements are set to 0. This means that an XML text editor would allow the user to introduce a value for a derived element. At parsing time however, this value is overridden by the output of the deriving function[2].

Previous attempts to extend XML Schema rest on the *redefine* construct [MCV04]. This means to extend the semantics of schema language construct with new behaviour. This accounts for a cleaner solution (i.e. more expressive) but, at the expenses of hindering schema sharing since partners are required to have their parsers similarly upgraded to account for the same *redefine* statements. Expressiveness of new concepts in XML Schemas and interoperability (i.e. the use of standards XML software) are two contrary requirements as it is reflected in the study of different approaches by Bernauer et al [BKK03].

By contrast, we resort to the `<xs:annotation>` element. Main benefit: allows designers to provide further information about elements without touching the schema language semantics. *Schematron* [JA01] illustrates this approach. Annotations may contain `<xs:documentation>` or `<xs:appinfo>` elements. The former is for human consumption whereas the latter provides instructions targeted at the

[2]Another alternative would have been to raise an error that indicates this situation.

processing application. Figure 2.1 shows an *<xs:documentation>* element which holds the author and the purpose of this schema. This information is intended for humans, hence an *<xs:documentation>* element is used. By contrast, derived element semantics are targeted at the XML parser, so they will be enclosed within *<xs:appinfo>* elements. The bottom line is that this solution permits to share the schema with other partners. Even if their parsers are not derivation-aware, they can still parse the standard part of the schema. This somehow preserves a kind of backward compatibility.

2.4.3 Examples of *XDerive* expressions

XDerive namespace is presented here through examples. All *XDerive* expressions, identified by *xd* namespace-prefix, are kept as content of the *<xs:appinfo>* element of the corresponding derived element declaration (i.e. *<xs:element>*) in the XML Schema.

Example 1. Deriving with local data: the *<subTotal>* element. The value of *<subTotal>* is calculated from the item's costs found in the order. Its specification using an *XDerive* expression is as follows:

```
<xs:element name="subTotal" type="xs:decimal" minOccurs="0">
   <xs:annotation>
      <xs:appinfo>
         <xd:derivingFunction>
            <xd:rule id="rule-1" test="//partialCost">
               <xd:documentation>
                  The subTotal is found adding all the partialCost of
                  the ordered items.
               </xd:documentation>
               <xd:action>
                  <xsl:value-of select="sum(//partialCost)"/>
               </xd:action>
            </xd:rule>
         </xd:derivingFunction>
      </xs:appinfo>
   </xs:annotation>
</xs:element>
```

A deriving function is described through its namesake element (i.e. *<xd:*

derivingFunction>[3]). Its content includes a set of rules that realize the function. A rule specification (i.e. *<xd:rule>* element) includes a description of its purpose (i.e. the *<xd:documentation>* element), the rule's antecedent (i.e. the *test* attribute) and the rule's consequence (i.e. the *<xd:action>* element). The *test* attribute holds an XPath expression, i.e. a predicate on the content of the instance document. In this case, the test checks that *<partialCost>* elements exist. The *<xd:action>* element comprises a set of standard XSLT instructions. The example uses *<xsl:value-of select="..."*> to obtain the value of the derived element using the *sum()* XPath function with the list of *<partialCost>* elements as parameter. The latest instruction **must be** *<xsl:value-of ...>* for simple type values and *<xsl:copy-of ...>* for complex type values. Notice that no prioritization schema has been included as only one rule is provided.

Example 2. Deriving with remote data and a value-based prioritization schema: the *<insuranceCost>* element. The value of *<insuranceCost>* is the maximum of the costs associated with the *category* of the distinct items included in the order. Its specification using an *XDerive* expression is shown in Figure 2.6.

This derivation policy implies more than one rule. If the order comprises items of distinct categories, more than one rule can be applied. The *<xd:prioritization Schema>* element indicates how to resolve this ambiguity. In this case, the designer chooses to combine the values returned by distinct applicable rules (held by the special *<actionResult>* element of the rule), and takes the highest one. This is supported by the *math:max()* function which is specified at the *combiningFunction* attribute. This attribute is held by the *<xd:valueBased>* element.

This example also illustrates the use of remote data. Here, the rule's antecedent checks the item's *category*. As this data is not available in the base document, the track document is consulted (referred to through the variable *$derivation Track*). The structure and generation of this document is postponed till section 2.5.1.1. Notice however, that the source of the external data should be indicated. The *<xd:source>* element serves this purpose. This element holds the name of the external deriving data (*derivingFact* attribute), the connection configuration (e.g. the database's URL, the driver and the like) to be used to retrieve this data

[3]The prefix *xd:* indicates the namespace where this element's tag is defined.

```
<xs:element name="insuranceCost" type="xs:float" minOccurs="0">
   <xs:annotation>
      <xs:appinfo>
         <xd:derivingFunction actuate="onLoad">
            <xd:documentation>The insurance cost will be the maximum among the
                   cost associated to each category of the ordered items.
            </xd:documentation>
            <!– rule-7 and rule-8 are omitted for the explanation –>
           <xd:rule id="rule-9" test="$derivationTrack//category[.='electronic']">
                <xd:documentation>If exist any item whose category is
                        'electronic' then the insurance cost will be 5.99 euros.
                </xd:documentation>
                <xd:action>
                   <xsl:value-of select="number(5.99)"/>
                </xd:action>
                <xd:source derivingFact="category" connection="itemInfo">
                   <xsql:query> SELECT category, ref FROM items
                                  WHERE ref IN $order/lineItem/ref</xsql:query>
                </xd:source>
             </xd:rule>
             <xd:rule id="rule-10" test="$derivationTrack//category[.='computers']">
                <xd:documentation> If exist any item whose category is
                        'computers' then the insurance cost will be 7.99 euros.
                </xd:documentation>
                <xd:action>
                    <xsl:value-of select="number(7.99)"/>
                 </xd:action>
                <xd:source derivingFact="category" connection="itemInfo">
                   <xsql:query> SELECT category, ref FROM items
                                  WHERE ref IN $order/lineItem/ref</xsql:query>
                </xd:source>
             </xd:rule>
             <xd:prioritizationSchema>
                <xd:valueBased combiningFunction="math:max($rule-7/actionResult,
                           $rule-8/actionResult, $rule-9/actionResult,
                           $rule-10/actionResult)"/>
              </xd:prioritizationSchema>
         </xd:derivingFunction>
      </xs:appinfo>
   </xs:annotation>
</xs:element>
```

Figure 2.6: The specification of *insuranceCost* using *XDerive*.

(*connection* attribute), and the SQL query that retrieves the value (*<xsql:query>* element).

Although our approach includes this connecting information as part of the schema of the base document, it is debatable whether this information should have been better recorded in the track document itself. The rationale for our decision is that the track document is a *generated* document, not a specification document. So all information should be included in the schema of the base document. A second aspect is modularity and locality. By associating the source of the deriving data together with the rule that uses such data, the removal/addition of rules is greatly facilitated. Even if this leads to repeating the source several times (if the same external data is used in distinct rules), the benefits outweigh this drawback. Moreover, the data is retrieve once from the database.

Example 3. Deriving with remote data and a rule-based prioritization schema: the *<deliveryTime>* element. The value of *<deliveryTime>* depends on the *shippingSpeed* and the *availability* of the items in stock. Its specification using an *XDerive* expression is shown in Figure 2.7.

This derivation policy also comprises several rules. However, unlike the previous example, only one rule is applied, even if the antecedent of several rules is met. That is, the prioritization schema is now rule-based. This implies that the system checks the rule's antecedents, keeps this result in a special element tagged *<antecedentResult>*, and finally, applies the priority rules. The latter resolves the conflict about which base rule to apply.

To this end, the *<xd:prioritizationSchema>* element now contains an *<xd:ruleBased>* element, which in turn contains the priority rules. Since a priority rule is a rule, its specification is similar to a base rule. But now the subject matter are the base rules. That is, a priority rule's test checks whether base rules can be applied or not, while its consequence indicates the base rule to be applied. Figure 2.7 gives an example.

The first priority rule in Figure 2.7 checks whether *rule-17* and *rule-18* are both met by the current order. If so, the priority rule resolves the conflict by applying *rule-17*[4]. Priority rules are evaluated sequentially till one is met. It is

[4]In the current version, the rule engine assumes a disjunctive test for the priority rules.

```
<xs:element name="deliveryTime" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:appinfo>
            <xd:derivingFunction actuate="onLoad">
                <xd:rule id="rule-16" test="shippingSpeed = 'Standard Shipping' ">
                    <xd:documentation>If the shipping speed is 'Standard Shipping'
                               then the delivery-time will be between 4 and 8 days.
                    </xd:documentation>
                    <xd:action>
                         <xsl:value-of select=" 'You order will arrive between ' +
                          date:sum($order/date,4)+' and '+date:sum($order/date, 8) "/>
                     </xd:action>
                </xd:rule>
                <xd:rule id="rule-17" test="count($derivationTrack//availability[
                               .='24 hours'])>3 and shippingSpeed='One Day Shipping' ">
                    <xd:documentation>If the shipping speed is 'One Day Shipping'
                               and there are available in 24-hours more than three items
                               then the delivery-time will be 2 days.</xd:documentation>
                    <xd:action>
                         <xsl:value-of select=" 'You order will arrive on ' +
                                                     date:sum($order/date,2)"/>
                     </xd:action>
                     <xd:source derivingFact="availability" connection="itemInfo">
                         <xsql:query> SELECT availability, ref FROM items
                                         WHERE ref IN $order/lineItem/ref</xsql:query>
                     </xd:source>
                </xd:rule>
                <xd:rule id="rule-18" test="shippingSpeed = 'Two Days Shipping' or
                                                    shippingSpeed = 'One Day Shipping' ">
                    <xd:documentation>If the shipping speed is 'One Day Shipping' or
                               'Two Days Shipping' then the delivery-time will be 4 days.
                    </xd:documentation>
                    <xd:action>
                         <xsl:value-of select=" 'You order will arrive on ' +
                                                     date:sum($order/date, 4)"/>
                     </xd:action>
                </xd:rule>
                <xd:prioritizationSchema>
                     <xd:ruleBased>
                         <xd:rule test="$rule-17/antecedentResult and
                                         $rule-18/antecedentResult">
                             <xd:action><xd:apply-rule select="rule-17"/></xd:action>
                         </xd:rule>
                         <xd:rule test="$rule-18/antecedentResult">
                             <xd:action><xd:apply-rule select="rule-18"/></xd:action>
                         </xd:rule>
                         <xd:rule test="true()">
                             <xd:action><xd:apply-rule select="rule-16"/></xd:action>
                         </xd:rule>
                     </xd:ruleBased>
                 </xd:prioritizationSchema>
            </xd:derivingFunction>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
```

Figure 2.7: The specification of *deliveryTime* using *XDerive*.

up to the rule designer to provide a coherent set of priority rules, i.e. rules whose antecedents are disjoint. To ensure completeness, a last rule is added in Figure 2.7 whose antecedent is always true. This accounts for the situation where none of the previous priority rules apply.

2.5 Artefact implementation: *XDerive*

XDerive is a notation for derived facts. That is *XDerive* is a knowledge model. This section addresses the execution model, i.e. the enactment of deriving function and its inclusion in current XML parsers. This section tackles the environmental requirements set in section 2.3.

2.5.1 How to enact *XDerive* expressions?

So far, we have addressed the knowledge model of deriving functions. Now, we need also to address how XDerive expressions are going to be enacted. In a database scenario, two answers are possible, namely, derived attributes are stored in the database or calculated on demand. The designer should balance: *i)* the additional cost of storing the derived data and keeping it consistent with the base data, with *ii)* the cost of calculating the derived data each time it is required [CB98].

Unlike derived data (as found in the database area), a business document should not be kept continuously consistent with the deriving data. Therefore, the *"calculated-on-demand"* option has not been contemplated. The content of derived elements are obtained as an aside process at parsing time. This process takes the *"base document"* as an input, and generates the *"derived document"* plus the "*track document*" (see section 2.5.1.1), where external deriving data is kept[5]. The next subsection introduces the "*track document*", called the *derivationTrack* document.

[5]Due to the static nature of business documents, this work does not contemplate the possibility of updating the derived element directly nor the updating propagation issue. The latter involves how modifications on the derived data are propagated to the deriving data. The challenge lies in the ambiguity of this transformation as the propagation is not always unique [CM89].

2.5.1.1 The *derivationTrack* document

A business document reflects an agreement between the distinct partners involved in the transaction. Paper-based documents usually have an attachment (e.g. conditions on sale) that describes the context in which this agreement is set. Such an attachment plays an important legal role in case of any demand or complaint about the transaction.

Likewise, "virtual documents" also need a similar attachment. This is particularly so in the presence of derived elements. Here, deriving data can be updated once the transaction occurs (e.g. the item price can be modified) so as to prevent the customer from recreating the context in which his order was processed. The solution envisaged here is to generate an attachment at the time the transaction (i.e. the order) occurs. Such an attachment holds a snapshot of the external deriving data which has been used to obtain the derived values. Such snapshots of the "business context" allow business partners other than the issuer, to be aware of the state of the business at the time the transaction took place so as to validate the appropriate application of the business policies. Such an attachment is generated as a by-product of the derivation process, and attached to the business document through an XML processing instruction. This is achieved by means of the *derivationTrack* document (see appendix B), a document which tracks the derivation process so that this derivation can be replicated at any moment by any of the partners involved. This document is automatically generated during the derivation process, and associated to the business document by means of the `<?xderive-derivationTrack ... ?>` processing instruction (see Figure 2.2, line 1).

Although such an approach can sound bizarre at first glance, it is similar to current practices to indicate the presentation of an XML document. In this case, how a document is rendered is separately described in an XSLT document. When the document is processed according to its "presentation document", the rendering is obtained.

Likewise, how a document has been derived can also be separately indicated in a distinct XML document: the *derivationTrack* document. When this document is processed according to its "track document", both the distinct business policies

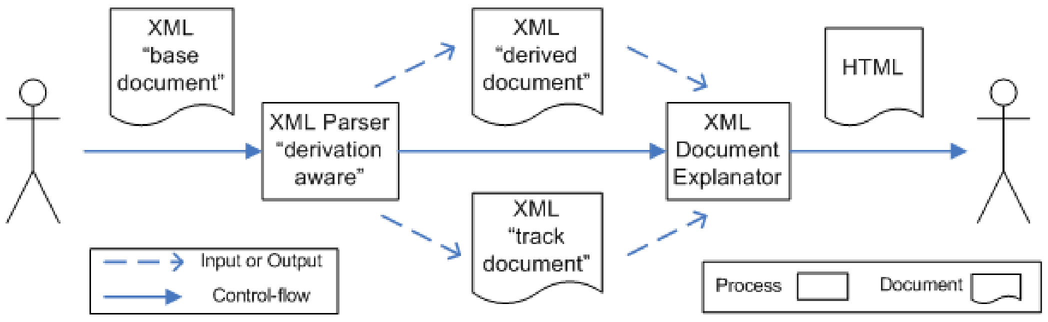


Figure 2.8: Document life-cycle: the activating, parsing and explanation stages.

and the consulted deriving values are made explicit.

The envisioned situation is depicted in Figure 2.8 where the following stages are contemplated:

1. ***activating stage***. The activator (e.g. either a customer or an application that sends the order automatically if the stock goes below a certain threshold) kicks off the whole process by submitting the "base document",

2. ***parsing stage***. At the issuer place, the derivation-aware parser takes the base document as an input and generates both the "derived document" and the "track document".

3. ***explanation stage***. Finally, either the activator, the issuer or the trusted third party, can process the "derived document" in explanation mode so as to ascertain both the policies and deriving facts that has been applied.

Therefore, the *derivationTrack* document records the values of the external deriving data (the *<xdt:derivingElements>* element), and the rules which have been applied during the derivation process (the *<xdt:derivedElements>* element). Appendix B gives an example.

The *derivationTrack* document keeps each external deriving data as an element. It is worth emphasizing that the parser only records the deriving data that is used in the derivation process. The first deriving element of the sample in appendix B follows:

```
<category ref="TV-1"> electronic </category>
```

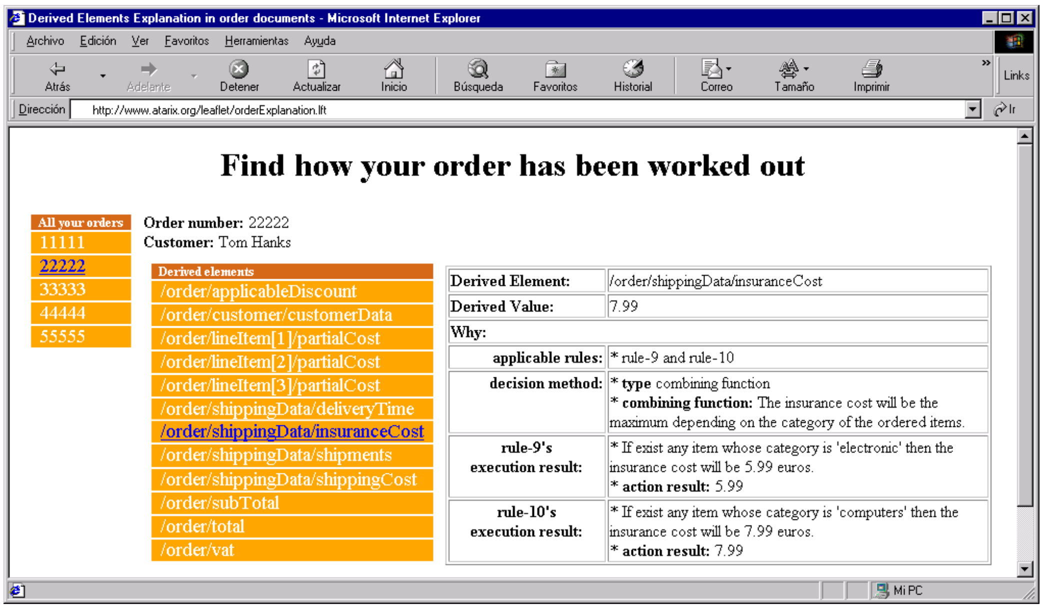
An attribute, named `ref`, is added to relate the element on the *derivationTrack* document with the associated element in the base document. Here, the attribute `ref` indicates that this is the category of the item *"TV-1"* to be found in the *order* document. Therefore, this work is based on the understanding that the base document should provide the means to univocally identify the entities participating in the transaction. In our running example, the base document should clearly indicate the customer and the ordered items. This is normally achieved through a reference or code. In our example, the customer is identified through the `<id>` element while an item holds a `<ref>` element.

Besides deriving elements, the track document keeps information about the derivation process through the `<xdt:derivedElement>` element. An example follows:

```
<xdt:derivedElement select="/order/shippingData/insuranceCost">
   <xdt:activated rules="rule-9 rule-10"/>
   <xdt:decision type="combining function" />
   <xdt:executed rule="rule-9">
      <xdt:actionResult>5.99</xdt:actionResult>
   </xdt:executed>
   <xdt:executed rule="rule-10">
      <xdt:actionResult>7.99</xdt:actionResult>
   </xdt:executed>
</xdt:derivedElement>
```

The data recorded includes: *i)* an Xpath to reference the derived element or attribute (`select` attribute); *ii)* which rules were activated (`<xdt:activated>` element); *iii)* which decision was taken, whether more than one rule were applicable (the `<xdt:decision>` element); and *iv)* which rules were finally executed (`<xdt:executed>` element). The above example states that *insuranceCost* was calculated after applying a combining function to the results of the rules *rule-9* and *rule-10*.

The rationale for having a *derivationTrack* document is to allow business partners to validate the conformance of the business transaction. As an example, consider an application that renders pending orders (e.g. similar to the one offered by *Amazon*). Partnership trustworthiness can be enhanced by allowing the customer not only to check the status and data about the order, but also ease the way for the customer to understand how the delivery time or insurance cost has been


Derived Elements Explanation in order documents - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás
Adelante
Detener
Actualizar
Inicio
Búsqueda
Favoritos
Historial
Correo
Tamaño
Imprimir
Links
Dirección
Ir
Mi PC

http://www.atarix.org/leaflet/orderExplanation.lft

Find how your order has been worked out

All your orders
11111
22222
33333
44444
55555

Order number: 22222
Customer: Tom Hanks

Derived elements
/order/applicableDiscount
/order/customer/customerData
/order/lineItem[1]/partialCost
/order/lineItem[2]/partialCost
/order/lineItem[3]/partialCost
/order/shippingData/deliveryTime
/order/shippingData/insuranceCost
/order/shippingData/shipments
/order/shippingData/shippingCost
/order/subTotal
/order/total
/order/vat

Derived Element:	/order/shippingData/insuranceCost
Derived Value:	7.99
Why:	
applicable rules:	* rule-9 and rule-10
decision method:	* **type** combining function * **combining function:** The insurance cost will be the maximum depending on the category of the ordered items.
rule-9's execution result:	* If exist any item whose category is 'electronic' then the insurance cost will be 5.99 euros. * **action result:** 5.99
rule-10's execution result:	* If exist any item whose category is 'computers' then the insurance cost will be 7.99 euros. * **action result:** 7.99

Figure 2.9: Browsing how derived elements have been calculated.

calculated. Currently, this involves the user reading through, potentially, several pages at the retailer's site. This is lengthy and cumbersome, and can put the user off. The rationale here is similar to the one found for the explanation mechanisms provided by some Expert Systems: improving the confidence of the user on the transparency and accuracy of the system. This is also so in a B2B setting.

Enhancing customer trustworthiness can be as easy as rendering the track document for the customer to replicate the derivation process. Figure 2.9 depicts a straightforward rendering of the track document shown in appendix B. An index of order numbers is shown on the left. Once an order has been selected, its track document is rendered. In this case, the system presents another index of the distinct derived elements found in the selected order. The user can then select one of these elements, and both the business policy and derived elements are shown on the right hand side.

More elaborate displays can be provided that allow partners to track, for instance, the number of orders that benefit from a certain offer, and in general, the number of cases where a certain business policy (realized as a deriving function) has been applied. In this way, partners can not only check the fulfilment of the agreements, but also whether these business policies have been beneficial.

2.5.2 Making validators *XDerive* aware

Document validation is a cornerstone of the XML world. On reception, partners check the validity of the receiving document against its schema (commonly described through XML Schema). This work proposes to extent XML Schema with deriving functions. Hence, the question arises about how validators are going to cope with this new kind of elements.

The `<xs:appinfo>` element allows the provision of additional information in the document schema. The questions are now who will and how to interpret this information. A possibility is to use a two-step approach whereby the semantics of the new constructs are first compiled into a run-time validator, -normally using *XSLT*. Next, this validator is used to check the compliance of the instance document against the new constructs.

This work proposes an alternative way. Rather than building a new validator that complements the *XML Schema* parser, the XML parser itself is extended to become "derivation aware". Applications can use the extended parser to parse the instance document as they did before except that now, the new constructs (i.e. *XDerive*) are also interpreted.

The main benefit is "parser transparency", that is, parser enhancements do not imply changes to the applications using the parser: the application ignores whether the processed documents contains the new constructs (in this case, derived elements) or not. Of course, this option is feasible only for open and modular parsers. However, such parsers have seemed to increase in popularity since the publication of JAXP (Java API for XML Parsing) [Sun]. The XML Parser of the Apache Project, Xerces2 [The01], and Oracle's XML Parser V2 for Java [Ora02], are examples of JAXP-based parsers. The latter has been used for this work.

JAXP advocates for configurable XML parsers. In its most basic form, an XML parser just generates a DOM tree for the input document. Further enhancements should be reflected through configuration parameters. Specifically, JAXP provides two initial configuration parameters, namely, *namespaceAware* and *validating* (see Figure 2.10). The former allows a parser to tackle documents where distinct namespaces are used, whereas *validating* indicates whether the parser should also check the conformance of the document against a DTD or XML

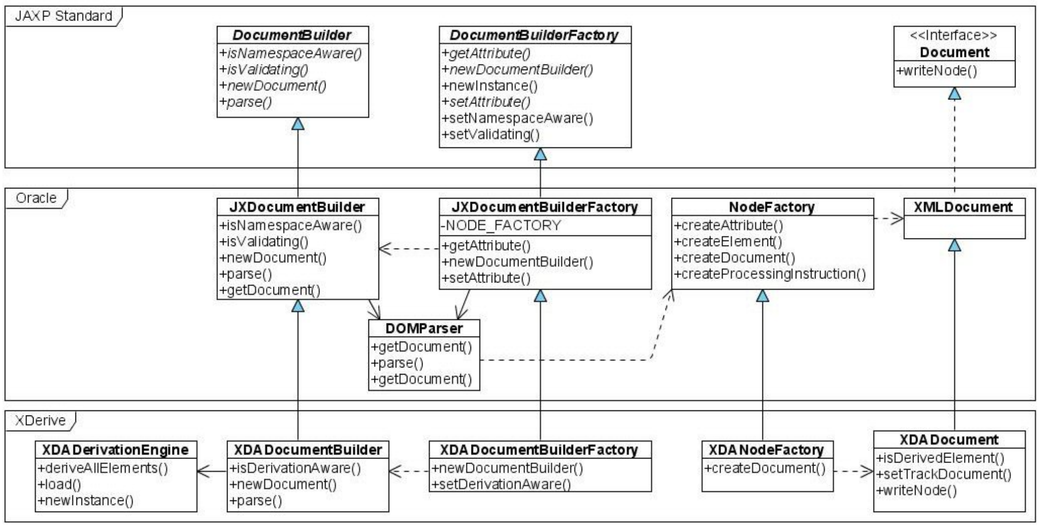


Figure 2.10: Extensions that make a JAXP-compliant parser "derivation aware".

Schema. Hereafter, companies can enhance their own XML parsers by providing additional features.

Aligned with this approach, this work is about enhancing an XML parser with derived elements. Thus, the parser has been extended with an additional configuration parameter, *derivationAware*, that allows parsers to face derived elements. The next subsections examine how this has been achieved.

2.5.3 An outline on the JAXP standard

The JAXP architecture follows the *factory method* pattern for the creation of both document builders and nodes[6]. These patterns are used when *"a class can't anticipate the class of objects it must create"* or *"a class wants its subclasses to specify the objects it creates"* [GHJV95]. The former situation is found for instantiating document builders whereas the latter arises for node creation. Figure 2.10 shows this situation.

The creation of document builders is handled through both the *DocumentBuilderFactory* and the *DocumentBuilder* classes. Both classes are abstract. *DocumentBuilderFactory* defines an abstract method, *newDocumentBuilder()*, that

[6]A document is realized as a set of nodes arranged in an arborescent way: the DOM tree.

knows when a new *DocumentBuilder* should be created, but it ignores what kind of *DocumentBuilder* should be created. This aspect is delegated to its subclasses (e.g. the *JXDocumentBuilderFactory* class) that redefines that abstract method to indicate the appropriate *DocumentBuilder* to be used (e.g. the *JXDocument-Builder* class). The *newDocumentBuilder()* method is called a *factory* method because it is responsible for "manufacturing" an object.

As for nodes, the *NodeFactory* class plays the *factory* role. Unlike the previous example, *NodeFactory* is a concrete class, that is, this class already provides a default implementation. This implementation is realized through the methods *createAttribute(), createElement(),* etc., one for each kind of node that can be found in an XML document. These methods can then be specialized to account for special requirements.

This architecture can now be completed by parser vendors. Figure 2.10 shows this extension for Oracle's XML Parser V2 for java. Specifically, the parser comprises the following classes:

- *JXDocumentBuilder* class. It is in charge of: 1) instantiating the XML DOM parser (implemented by the *DOMParser* class); *2)* starting the parsing process; and *3)* instantiating a DOM Document object to build a DOM tree (through the *newDocument* method). This class realizes the *Document-Builder* abstract class of JAXP.

- *JXDocumentBuilderFactory* class. It instantiates a *JXDocumentBuilder* object. Once the *JXDocumentBuilder* instance creates the *XML DOM* parser, the factory configures the just created parser according to a previously established configuration. As an example, consider the way to set a parser to be "validation aware". To this end, the factory has a *setValidating()* function to set this feature. If this property is set to "true", the parsers generated from then on will be "validation aware". This class realizes the *DocumentBuild-erFactory* abstract class of JAXP.

- *DOMParser* class. An object of this class is the XML parser as such. It takes an XML document as an input, and generates a DOM tree in memory, by means of the *NodeFactory* class. This process is lead by the *parse*() method. The DOM tree is an instance of the *XMLDocument* class.

- *NodeFactory* class. This class is responsible for creating different DOM objects during the parsing process. Hence, distinct methods are available to tackle each of the node types available in XML: *createAttribute(), createElement(), createDocument()* and the like. This allows one to attach application-specific semantics by instantiating application-specific classes.

These classes support an XML DOM parser. Our aim is to make this parser "derivation-aware".

2.5.4 Making the parser "derivation aware"

A JAXP parser is "derivation aware" if it *1)* interprets deriving functions (specified through *XDerive*) and *2)* generates the track document as a by-product of the parsing. To this end, the previous classes have been extended as follows. First, the *XDADocumentBuilder* class extends *JXDocumentBuilder* class in order to specialize an XMLparser in derived data. Besides parsing the "*base document*", this subclass also give support to interpret the deriving functions, and generate both the "*derived document*" and the "*track document*".

Second, a new configuration parameter (i.e. *derivationAware*) and its setting method (i.e. *setDerivationAware*()) are added by the *XDADocumentBuilderFactory* class, which extends the *JXDocumentBuilderFactory* class' functionality. If this parameter is set, the factory will create an instance of *XDADocumentBuilder* rather than a *JXDocumentBuilder* object.

Finally, the *NodeFactory* class is extended by the *XDANodeFactory*. Besides parsing the document, an *XDADocumentBuilder* invokes the *XDADerivationEngine* when a derived element is found. This class supports the interpreter of XDerive elements. Moreover, the *XDADerivationEngine* simultaneously records the rules being applied, building up the track document. This means that now the parser has to cope with two documents: the derived document and the track document. *XDANodeFactory* specializes the *createDocument()* method so as to handle the two documents simultaneously. The rest of the behaviour (i.e. the creation of the distinct node types available in XML) is left untouched. The specialized *createDocument()* constructs a DOM tree which can hold another DOM tree. The first is for the derived document, and the other is for the track document. The track document is attached by the *setDerivationTrack()* method. And finally, when the document is saved, both documents are linked by adding the `<?xderive-derivationTrack ...?>` processing instruction to the derived document. This is achieved by the *writeNode()* method (in the *XDADocument* class).

Figure 2.11 depicts an interaction diagram that reflects the "derivation aware" parser at work:

- Firstly, the *Client* application configures the *DocumentBuilderFactory* to create a "derivation aware" XML DOM. To this end, the application creates a new instance of *XDADocumentBuilderFactory* and sets the corresponding configuration parameter, namely, sets *XDANodeFactory* as the value of the parameter *NODE_FACTORY* (using the *setAttribute()* method) and assigns "true" to the *DerivationAware* parameter (using the *setDerivationAware*() method). After this, an instance of *XDADocumentBuilder* is created which in turn, creates a new instance of a *DOMParser*.

- Secondly, once the parser is in place, the *parse*() method of *XDADocumentBuilder* is invoked to initialize the parsing process of the given document. This requirement is passed over to the *DOMParser*. The *DOMParser* creates both the document and the elements, which are appended to the document.

- Next, the *XDADocumentBuilder* retrieves the created document. If this document has an attached schema then, a new instance of *XDADerivationEngine* is created. The derivation engine loads the schema and compiles the deriving functions into an XSLT document (using the *load()* method).

- If there exists derivation rules and this document has not been parsed yet (i.e. no *track document* is attached) then, the *deriveAllElements*() method is invoked. This method both calculates and appends all derived elements. Then, the *XDADerivationEngine* passes the track document to the *XDADocument* instance.

- Finally, the application saves the parsed document by invoking the *write Node*(*aURL*) method. This method attaches the *derivationTrack* document

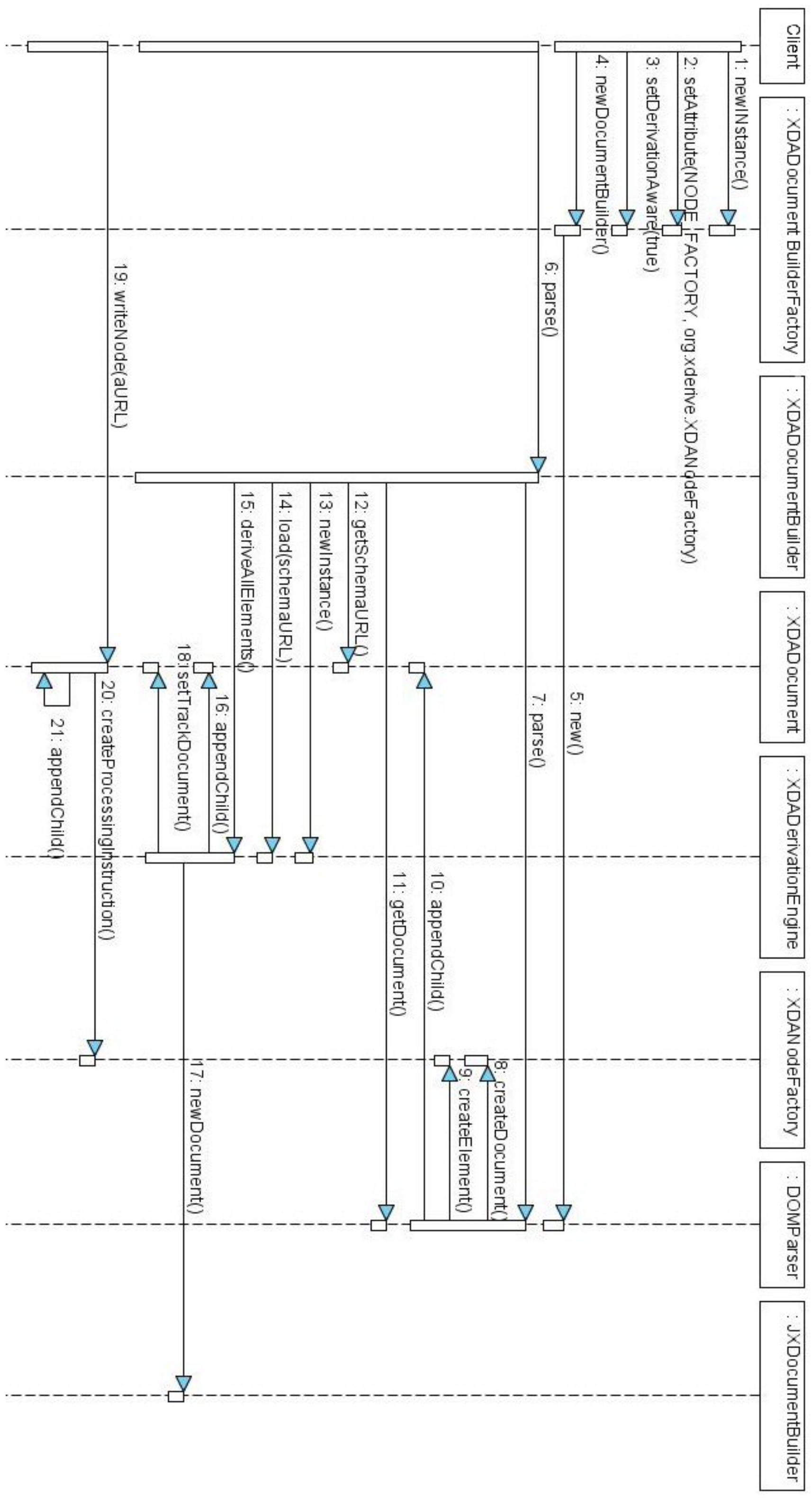


Figure 2.11: A “derivation aware” XML parser at work.

to the derived document by appending the processing instruction `<?xderive-derivationTrack ...?>`. This instruction holds the URL where the track document is stored (see Figure 2.2). Finally, *writeNode* saves both documents in the given URL.

2.6 Demonstrate artefact

This section aims to demonstrate the feasibility of using annotations for XML Schema extensions for the special case of deriving functions. To this end, a XDerive construct is designed together with an execution model, and the corresponding engines that serve as a proof of concept. Besides the accommodation of deriving functions into XML Schema, another question is whether XDerive is expressive enough or not. Unfortunately, we were not able to check it out with a real life case, and hence resorted to the case study presented in subsection 2.5. Though this hinders external validity, it serves to demonstrate a richer set of cases that would have been possible if sticked to a real situation.

2.7 Related Work

Active documents. Derived documents are closely related to active documents [SB02, ABM$^+$04, BCP01, NJB03]. An active document is a document that has a related behaviour that reacts to events occurred around the document. Active documents systems functionality is similar to active database systems [PD99]. The main similarities and differences of our work with others are:

(i) the *purpose*, our approach derives a document including some data that is derived from other data: *local* or *remote* (similar to [ABM$^+$04, NJB03], whereas [SB02, BCP01] change a document data in case that some other data (in other document or database) has changed.

(ii) active events, our approach reacts to the use of a document (i.e. *load* or *query* data events), whereas [SB02, BCP01] reacts to modifications (i.e. *insert*, *delete* or *update* events) in other documents. However, [ABM$^+$04] introduces explicit web-service calls in the instance document.

(iii) expressiveness and *interoperability*. Our work agree with most of them (except [BCP01] that uses directly XSLT templates) that active behaviour description needs a specific vocabulary in order to be more expressive. Moreover, our work promotes XML Schema extension as Active XML Schemas in order to reuse and gain interoperability.

There are active document system in the area of expert systems, which are more versatile and it performs analysis that are presently outside this work. For example, ADF[Zhu03] focus on services like *complex question answering*, that requires search, retrieval, reasoning and browse engines.

Rule Markup languages. The Rule Markup Initiative is striving to define *"an open, vendor neutral XML-based rule language standard ... permitting both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks"*. The current version of RuleML[7] 0.86 (in June 2004) only covers a very limited form of derivation rules, similar to Horn clauses [HBD+04]. This however, would be expressive enough to support derivation rules (except meta-rules for conflict-resolution) as the ones used in this work. In what follows, we highlight the main differences between using *RuleML* and the approach described here.

A *RuleML* rule contains facts, implications and queries. It is then a complete description of a production system as found in the Expert System realm. The *query* states a hypothesis to be proved by applying the *implications* which finally rest on the *facts*. In our approach, facts and implications are decoupled: facts are found in the business document whereas implications are described in its schema. Hence, our approach is better aligned with the database way. That is, databases make a clear distinction between the intension (i.e. the schema) and the extension (e.g. the tuples) where each aspect greatly differs in size and volatility.

Another important difference stems from the facts being XML business documents. Rather than using a Prolog-like approach, our rule's antecedent is an XPath expression that checks the existence of some data in the XML document. This leads to more legible expressions but also makes these expressions dependent on the document structure.

[7]http://www.ruleml.org

Derived data in database systems. For example, STRIP is a main memory resident soft realtime database system implemented at Stanford. On top, the STRIP rule system provides a unique transaction facility which accounts for very efficient incremental maintenance of derived data. The two benefits provided by unique transactions are: *"(i) they allow rule actions to act on database changes batched across transaction boundaries not just within one transaction; and (ii) they allow the batches to be partitioned in any way that reduces the cost of the derived data computation"* [AGMW97].

2.8 Conclusions

This chapter provides some insights on how *XML Schema* can be extended with derived elements. XDerive is proposed. Understandability, interoperability, completeness and expressiveness have been the guiding principles throughout. Understandability have been improved since XDerive expressions are more declarative that its code counterparts. Interoperability is protected by resorting to the *annotate* construct. Completeness is addressed through the track document which records the state and business policies used at the time the transaction occurs. Finally, expressiveness is checked out through a far from trivial case study.

Parts of the work described in this chapter have been previously presented:

- O. Díaz and F. I. Anfurrutia. Improving self-interpretation of XML-based business documents by introducing derived elements. Electronic Commerce Research and Applications (ECRA), 4:264–282, 2005.

- F. Ibáñez, O. Díaz, and J. J. Rodríguez. Extending XML Schema with Derived Elements. In *Proceedings of the IFIP WG8.1 Working Conference on Engineering Information Systems in the Internet Context*, volume 231 of IFIP Conference Proceedings, pages 53– 67. Kluwer Academic Publishers, 2002

Chapter 3

XML in Software Product Lines

3.1 Overview

Software Product Lines (SPLs) is a popular approach among Software Factories. The challenge: managing a large but similar set of products (a.k.a. a product family). *Feature-Oriented Programming* (FOP) is a paradigm to support SPLs whereby core artefacts are gradually enriched with features till the desired product is obtained. Implementation wise, enrichment is achieved through composition, i.e. core assets are said to be composed with the code of the feature (a.k.a. delta). This chapter elaborates on the notion of XML composition by rising three issues: the unit of composition, the ways of composition, and the laws of composition when code artefacts are realized as XML documents. A vocabulary is introduced for defining XML deltas. So-defined artefacts can next be validated and composed to output enhanced XML documents.

The chapter is organized along common Design Science activities: explicate problem, define requirements, artefact design, artefact implementation and artefact demonstration. Related work and conclusions end the chapter.

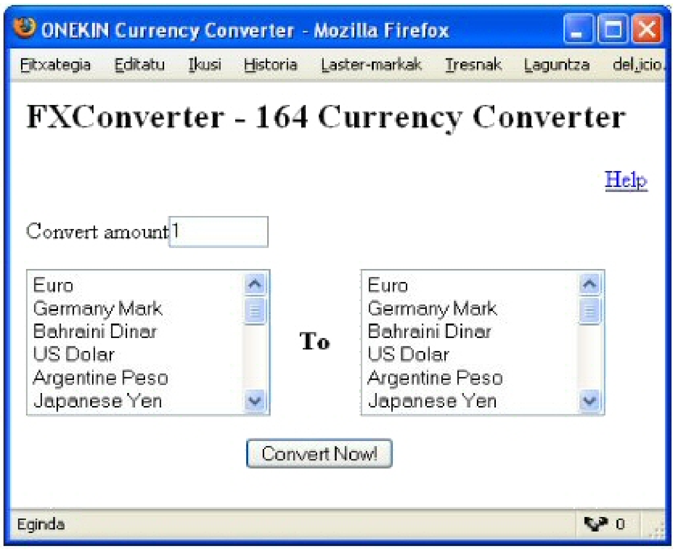


Figure 3.1: The base feature for CuCoWA.

3.2 Problem Explanation

3.2.1 In which practice does the problem appear?

Software Product Lines (SPLs) offer a paradigm to develop a family of software products. The focus shifts from the development of an individual application to the development of *core assets* that are used to develop a family of applications [CN01, PBvdL06]. Core assets are artefacts to be used in the production of more than one product. Products (i.e. software applications) within a family differ in the features they support. And feature realisation frequently crosscuts distinct core assets. Hence, SPLs should consider how core assets are engineered for facing *the variability* that goes with the presence or absence of the distinct features the SPL supports. Therefore, the management of variability is a cornerstone in SPLs.

An approach to variability management is *step-wise development* (SWD). This paradigm develops complex program out of simple programs by incrementally adding details [Wir71]. One realisation of SWD is *Feature-Oriented Programming* (FOP) which aims at large-scale compositional programming and feature modularity in SPLs [BSR04]. *FOP* departs from current “clone&own” practises by leveraging reuse of the common parts, and separating variable and changing

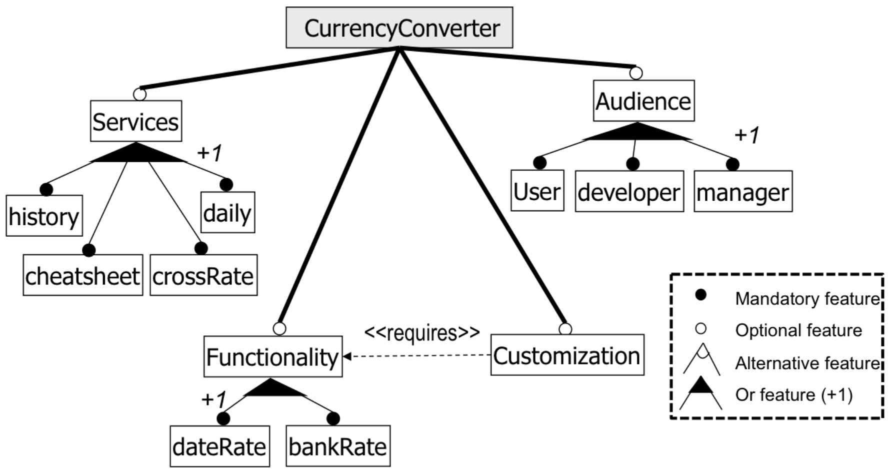


Figure 3.2: Feature model for *CuCoSPL*.

parts as *program deltas*. The final product is obtained through composition: the common parts (i.e. core assets) are leveraged with the program deltas that realise the variations for the product at hand. As an example, consider a *Currency Converter Web Application (CuCoWA)*. This application facilitates information about converting distinct currencies[1]. Figure 3.1 provides a screenshot of such application. Basically, the user sets a source and a target currency and the application provides the conversion for a given quantity. The issue arises when this base functionality varies. Figure 3.2 depicts a feature model that portrays different variations on how CuCoWA products might look like. Unlike a CuCoWA product, the CurrencyConverter SPL supports a product family, i.e. a set of CuCoWA products whose variations are expressed by the SPL's feature model. This diagram outlines how the base core can be enhanced by adding:

- *currency services*, which admits the following variants: *cheatsheet*, obtains the conversion of the most used amounts of money; *history*, obtains the historical exchange rate for any currency pair; *crossrate*, generates a currency cross-rate table; *daily,* allows to obtain a multi-currency table of currency exchange rates,

- *dateRate,* which allows to introduce a date to make the currency conversion with the rates at the given date,

[1]For a working example see *www.oanda.com/convert/classic.*

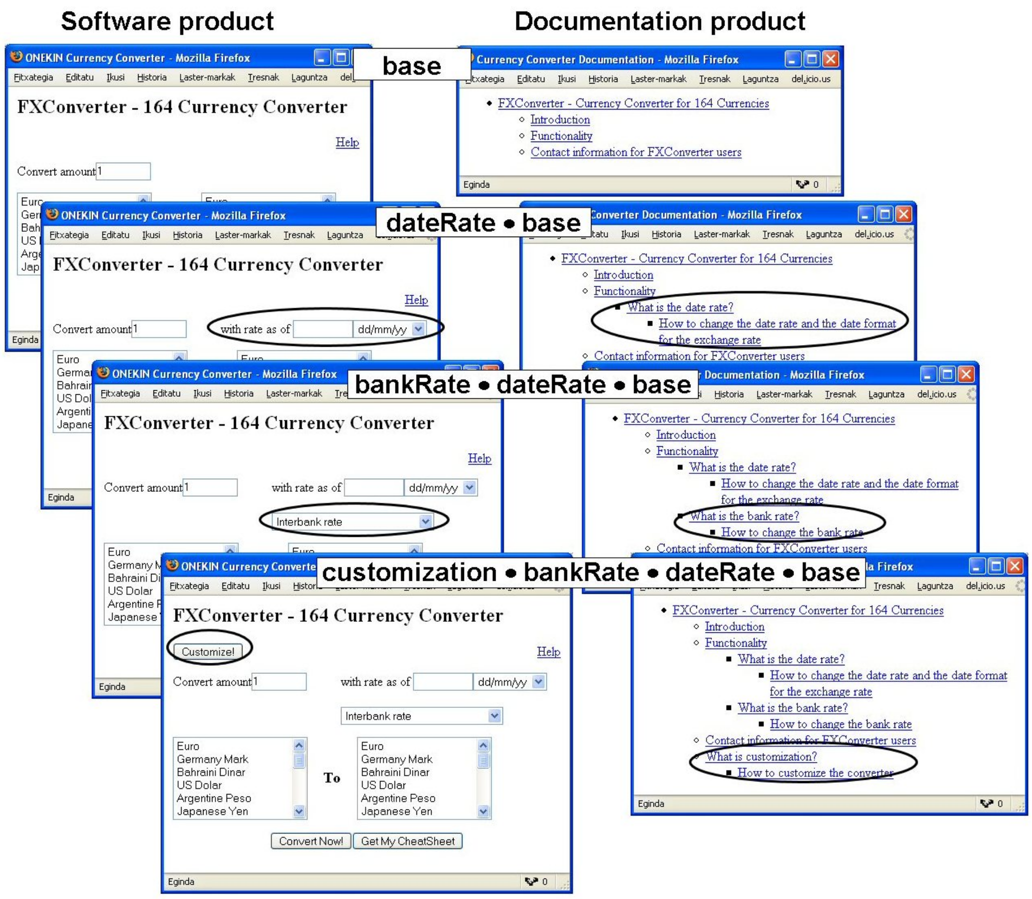
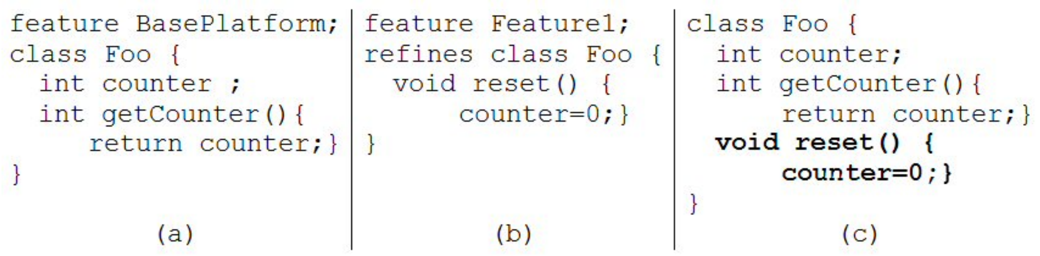

Software product
Documentation product
base
dateRate • base
bankRate • dateRate • base
customization • bankRate • dateRate • base
FXConverter - 164 Currency Converter
Help
Convert amount
with rate as of
dd/mm/yy
Interbank rate
Customize!
To
Convert Now!
Get My CheatSheet
FXConverter - Currency Converter for 164 Currencies
Introduction
Functionality
What is the date rate?
How to change the date rate and the date format for the exchange rate
What is the bank rate?
How to change the bank rate
Contact information for FXConverter users
What is customization?
How to customize the converter

Figure 3.3: The *CuCoWA* product family

- *bankRate*, which permits to include banking fees,
- *customisation*, which allows users to personalise the application by providing default values for the *sourceCurrency*, the *targetCurrency* and the preferred *currencyList* properties. Moreover, these options can be extended with the *defaultDateFormat* and/or *defaultBankRate* properties depending on whether the *dateRate* and/or *bankRate* features are selected, respectively.
- *audience,* which captures documentation variability based on the targeted reader.

FOP promotes a way to SPL development whereby a complex application is developed from a simple application by adding features incrementally using function



```
feature BasePlatform;
class Foo {
  int counter ;
  int getCounter(){
      return counter;}
}
```

(a)

```
feature Feature1;
refines class Foo {
  void reset() {
       counter=0;}
}
```

(b)

```
class Foo {
  int counter;
  int getCounter(){
       return counter;}
  void reset() {
       counter=0;}
}
```

(c)

Figure 3.4: Refinement of a class: a) *base* feature; b) *Feature1*; c) *Feature1•base* composition.

composition: For instance, the expression:

dateRate • *base*

denotes a product that is synthesized by composing the *base* with *dateRate* where both *base* and *dateRate* denote the set of artefacts that realized the namesake functionality. A more complex example would be:

cheatsheet • *crossrate* • *daily* • *customisation* • *dateRate* • *base*

This expression (a.k.a. configuration equation) accounts for a more complex SPL product. However, the *modus operandi* is just the same: composing deltas which gradually enrich a base core with a pre-set list of features. Figure 3.3 shows a set of CuCoWA products and how they have been gradually enriched.

Feature models serve to communicate the SPL variability. But how are feature realized? How are features mapped into code? Ideally, this mapping should be 1-to-1, i.e. one feature is realized through a code artefact. This is the vision of the Feature-Oriented Paradigm [ABKS13]. Features are realized using a mixin-like mechanism (a.k.a. refinements). Figure 3.4a shows a base artefact Foo defining variable members (*x* and *y*), and methods (*getX* and *getY*). This base artefact can now be incrementally extended by adding a new method reset() that extends the functionality of the base with a new feature *Feature1*. Figure 3.4b shows such extension using the Jak language [BSR04]. The expression *Feature1•Base* returns a Java artefact which holds feature *Feature1* (see Figure 3.4c). Likewise, *Feature2* •*Feature1•Base* stands for the *base* being enhanced with features *Feature1* and *Feature2*, where the order of feature composition (i.e. from right to left) can matter.

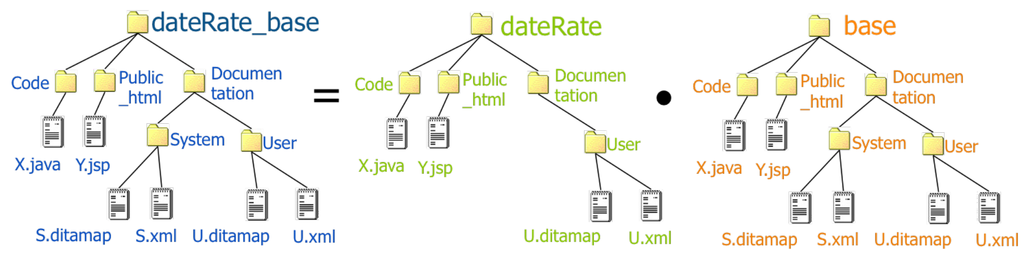


Figure 3.5: *JDeveloper* directory structure: *base* and *dateRate* feature implementation.

At first sight, this resembles regular inheritance. Notice however, that *Feature1* is not realised through a subclass of *Foo*. Rather, the very same class *Foo* is being extended. There are not two classes but a single class that is being incrementally extended to hold a new feature. Furthermore, the class being extended is not fixed at compile time (like in regular inheritance) but decided at composition time. In this way, a feature function behaves like a mixin inheritance, i.e. a class whose super class is parametrised. Since the super class is not fixed until composition, distinct feature functions on different (and unpredictable) order may be composed to yield a class.

However, a software product is more than a set of classes. A product is multifaceted, i.e. it admits multiple representations (e.g., Java classes, testing cases, SQL scripts, HTML pages, configuration files, etc.). And a feature can need to leverage one or several of those representations to inlay the desired functionality into the base program. Therefore, a feature realization is a composite, i.e. a unit of enhancement (i.e. a functional increment) that can potentially affect distinct representations. Some features can affect the rendering side of the product (e.g. realised as HTML artefacts), others the configuration side (e.g. realised as an XML document), and yet others can affect all of them. Thus, a feature function *f* encapsulates a set of artefacts *af* , *bf*, and *df*, we write *f= {af , bf , df }*. Similarly, *i = {ai, bi, ci}* says that feature *i* encapsulates artefacts *ai, bi* and *ci*. As artefacts themselves may be sets, a feature is a nested set of artefacts. Directories can be used to represent nested sets. This vectorial representation of features leads to “deep composition” whereby composition is nestedly and gradually applied to the affected artefacts (see Figure 3.5 as an example). Hence, adding a feature to a program refines each of the program’s representations, e.g.

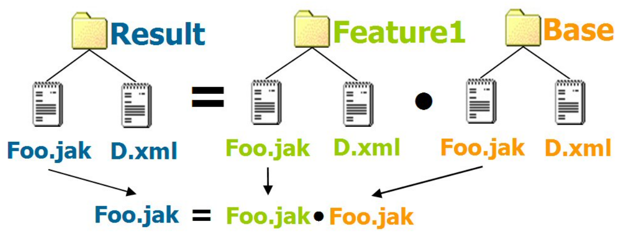


Figure 3.6: Composing features as directories.

$$i \bullet f = \{ai, bi, ci\} \bullet \{ af, bf, df\} = \{ ai \bullet af, bi \bullet bf, ci, df \}$$

where artefacts with the same name (ignoring subscripts) are composed pairwise and the new ones are simply added.

Figure 3.6 provides an alternative representation of the composition of features *Base* and *Feature1* where directories are folded together by composing corresponding artefacts in each directory. The result is feature *Result*, where artefact *Foo.jak* of *Result* is synthesised by composing *Foo.jak* (from *Feature1*) with *Foo.jak* (from *Base*).

The polymorphism of the • operator is central to feature composition. Artefacts of a given type (*.jak*, *.b*, etc.) and their refinements are defined in a type-specific language. That is, the definition and the refinements of *.jak* files are expressed in the *Jak*(arta) language, a superset of Java [BSR04]. The definition and refinement of *.b* files are expressed as *Bali* grammars, which are annotated *BNF* files. And so on. One or more tools implement the • operator for each artefact type. AHEAD Tool Suite (AHEAD TS) encompasses a set of tools that analyse and compose artefacts using refinements. AHEAD TS holds the distinct artefacts of the SPL, and produces a SPL product out of a feature equation (e.g. *Feature2•Feature1•Base*). So far, AHEAD composition is limited to Java and BNF artefacts.

Therefore, we address the following problem:

the lack of refinement and composition mechanisms for XML artefacts when in a feature-oriented programming setting

3.2.2 What is the problem and the negative consequences of not addressing this problem (or its benefits)?

The rationales for bringing FOP to the XML realm include:

- increasing presence of XML artefacts. XML artefacts play a preponderant role in current software practises, specially in the Web setting. XML can be found in both code artefacts (e.g. rendering markup languages such as XHTML[AME00], control-flow configuration files described through Struts [Theb], build process configuration files such as Ant [Thea], etc), and non-code artefacts (e.g. deployment descriptors such as *portlet.xml*[JCP03], UML diagrams serialised through XMI[Obj], asset documentation through RAS[Obj05], etc.).

- limited reused practices for XML artefacts. Studies revealed that the cloning rate of web-specific artefacts (e.g. mainly XML files) was considerably higher than general artefacts (e.g. *Java*, *C++*, etc) within web applications of the same organization [RJ05].

In an attempt to tackle reuse in the Web, approaches have been made to bring object-orientation (OO) and componentisation to the HTML realm [GWG97], [SWGZ00], [KNC01]. However, OO does not provide the right level of granularity. Java is a case in point. Mechanisms have been devised for driving Java towards FOP by introducing the notion of **delta** as a coarse-grained semantically meaningful construct on top of Java classes (Batory's Jak language [BSR04]). Likewise, XML will need similar amendments for the FOP benefits to reach the XML realm.

The vision is for XML artefact to be conceived incrementally: core assets are gradually enhanced with **deltas** (i.e. features) till the desired product is **synthesised**. This brings FOP benefits, i.e.

1. enhanced reusability of XML artefacts since commonalities and variabilities can be defined separately,

2. flexibility in the selection of variable content and their composition,

3. decoupling validation of XML core artefacts from XML deltas.

It is most important to note that the nature of the refinement depends on the artefact being refined, e.g. Java artefacts are not necessarily refined in the same way that XHTML artefacts. When the artefact is *".java"*, a class refinement can introduce new data members, methods and constructors to a target class, as well as extending or overriding existing methods and constructors of that class [BSR04].

But, what is meant to refine an XML artefact? The answer can not just mimic solutions for other programming paradigms but should be tuned to the XML way of programming. In the light of this observation, we pose the following research question:

- *how to introduce deltas into XML artefacts?*
- *how are XML products synthesized out of deltas?*
- *how are XML deltas defined, composed and validated?*

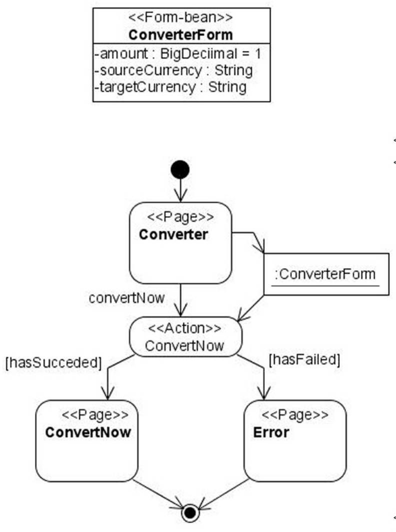
3.3 Requirement Definition

Our proposal to address the aforementioned issues is providing an XML vocabulary for delta definition. Main requirements include: expressiveness and suitability.

3.3.1 Expressiveness

Expressiveness refers to the degree to which a set of constructs is capable of representing the entities of interest in a domain. This subsection introduces the expressiveness requirements with the help of an example.

Let's go back to the CuCo SPL. CuCo's products are J2EE application [SSJ02], implemented using Apache Struts [Theb]. Struts follows the MVC pattern [KP88] where the Controller isolates the control-flow between the Model and the View. Let us focus on the Controller. Figure 3.7 (a) depicts the design of the ***base*** control-flow. The *Converter* page contains an instance of the form class named



(a)

```
<struts-config>
  <form-beans>
    <form-bean name="ConverterForm"
                  type="org.apache.struts.action.DynaActionForm">
      <form-property name="amount" type="java.math.BigDecimal"
                        initial="1"/>
      <form-property name="sourceCurrency" type="java.lang.String"/>
      <form-property name="targetCurrency" type="java.lang.String"/>
    </form-bean>
  </form-beans>
  <action-mappings>
    <!--Page States-->
    <action path="/Page.Converter.display" forward="/Converter.jsp"/>
    <action path="/Page.ConvertNow.display" forward="/ConvertNow.jsp"/>
    <action path="/Page.Error.display" forward="/Error.jsp"/>
    <!--Page Events-->
    <action path="/Page.Converter.events" name="ConverterForm"
              type="org.oneking.util.struts.action.DispatcherAction">
      <forward name="convertNow" path="/Action.ConvertNow.do"/>
    </action>
    <!--Action Activities-->
    <action path="/Action.ConvertNow" name="ConverterForm"
             type="org.onekin.xak.example.struts.actions.ConvertNowAction">
      <forward name="hasSucceeded" path="/Page.ConvertNow.display.do"/>
      <forward name="hasFailed" path="/Page.Error.display.do"/>
    </action>
  </action-mappings>
</struts-config>
```

(b)

Figure 3.7: **Base** control-flow: an UML diagram (a) and its code counterpart as the *struts-config.xml* configuration file (b).

ConverterForm. This form instance passes to the server the *amount* to be converted from the *sourceCurrency* to the *targetCurrency* for making a simple conversion, which is fulfilled by the *ConvertNow* action. Depending on the outcome of this action, distinct pages can be returned back: *ConvertNow*[2] (the *ConvertNow* action is executed successfully) or *Error* (if the previous action has failed).

This flow is embodied through the *struts-config.xml* along the following directives:

- Each activity action with stereotype *<<Action>>* in the UML diagram (see figure 3.7(a)) becomes an *<action>* element in the XML document (see figure 3.7(b)). The *<action>* basically maps the logical name of the action (the *path* attribute) to its physical realisation as a Java class method (the *type* attribute). The action's outcome determines the next step in the flow. A *<forward>* element is defined for each outcome (the *name* attribute) and its corresponding follow-on (the *path* attribute).

[2]Note that the *ConvertNow* id has been used to identify both an action and a page, which are differentiated by a stereotype.

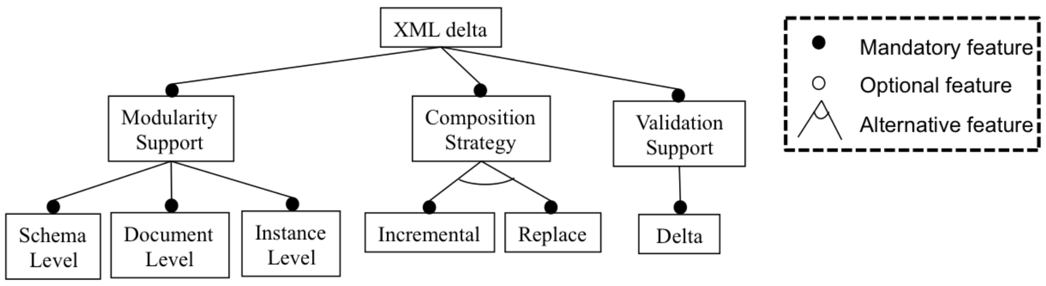


Figure 3.8: Expressiveness requirements.

- Each activity state with stereotype `<<Page>>` results into two kind of `<action>` element in Struts (see figure 3.7(b)): one for the display setting (e.g. *Page. Converter.display*), and the other for determining the next step depending on the event happened in the page (e.g. *Page.Converter.events*). The former maps the logical name of the page (the `path` attribute) to its counterpart physical page (the `forward` attribute). The `forward` attribute holds an HTML or JSP page which is next rendered to the user. The latter action describes a dispatcher action based on the output transitions from a page to any other page or action. Each transition is described using a `<forward>` element, as described before. Besides actions, the controller can also hold `<form-bean>` elements to indicate form properties[3].

Now the real meat of the FOP approach. This *base* artefact can now be enriched with features. Features are implemented in a similar way to the *base*. For our sample case, features might refine either the model, the view or the controller. That is a feature realization might be a compound of three files:

- *struts-config.xml*, if the controller is refined,
- dateRateConverterForm.java, if the model is refined,
- *dateRateForm.jsp*, if the view is refined.

The challenge is to incorporate these features **incrementally**. That is, starting with a *base* and next, apply deltas to progressively add refinements to this base. The

[3]Since forms are declaratively described and automatically generated, the developer does not need to implement an *ActionForm* class with *getter* and *setter* methods. Rather, the framework is responsible to convey form values from the request to the action.

question is how to define these refinements. Figure 3.8 shows the main requirements: modularity support, composition strategy and validation support. Next, we provide an overview of the rationales. How these requirements are finally met is addressed in Section 3.4.

Modularity support. We aim at constructing XML documents incrementally through composition. This begs the question of what is the appropriate granularity for this composition. The unit of refinement (i.e. a module) can be *i)* the document itself (i.e. *document-level*), *ii)* nodes inside a document (i.e. *instance-level*), and iii) nodes that belong to certain node types within the document (i.e. *schema-level*).

Composition Strategy. A *refinement* can introduce new data members, methods and constructors to a target class, as well as extend or override existing methods and constructors of that class [BSR04]. Likewise, *XML delta* can add, extend or override existing module elements. This requires: *(i)* a namespace to indicate which elements play the role of modules and their delta, and *(ii)* to realise the operational semantics of composition for XML modules by overloading the composition operator $\bullet$.

Validation Support. SPLs last longer and involve larger teams than those of single products. Specifically, deltas might well be defined separately for different engineers at different times along distinct release agendas. Therefore, the information available at delta-definition time basically includes the Feature Model and the *base*. Domain engineers should not make any assumption about how other deltas are realised, except those explicitly capture in the Feature Model. Nevertheless, deltas should smoothly fit together when synthesising the SPL products. In other words, mismatches at product-synthesize time should be minimised.

3.3.2 Suitability

Suitability has been defined as "the degree to which an artefact is tailored to a specific *practice*, focusing only on its *essential* aspects (also called inherence or precision)". By "practice" we understand the setting in which the outcome is going to be used. In this case, this setting is the XML realm. This in turn begs the question of what is the essence (as opposed to the accident) of XML programming.

This is our perspective:

- XML introduces a self-describing approach to data representation where the data is accompanied by a label (a.k.a tag) that describes the data. Data is structured in a tree-like way along two main constructs: elements and attributes (hereafter, commonly referred to as "nodes"). Therefore*, **XML deltas should be defined in terms of inserting or replacing nodes.***

- XML documents should frequently conform to a schema. The process of checking this out is known as validation. Validation plays a core role in XML programming. Therefore, ***validation should be applied to deltas.***

3.4 Artefact Design: XAK namespace

Previous section sets the requirements for a potential solution to the problem of extending FOP to XML artifacts This section introduces XAK (pronounced "sack"), an XML Schema namespace (a.k.a. vocabulary) for delta description. Based on the aforementioned requirements, XAK should be designed in a way that facilitates validation and composition of XAK-aware documents. This introduces three issues: the unit of composition, the ways of composition and the laws of composition.

3.4.1 The unit of composition

We aim at constructing XML documents incrementally through composition. This begs the question of what is the appropriate granularity for this composition. This subsection delves into this issue.

3.4.1.1 The first approach: node-based modularization

A first approach could be to consider any XML node as the unit of composition. Under this assumption, any element or attribute of the *struts-config* vocabulary can be refined, whatever this means. For understanding sake, let us introduce an XML Refinement language using the XR namespace.

```
<xr:refines xmlns:xr="http://www.onekin.org/XRefine"
            artefact="struts-config.xml" feature="dateRate">
   <xr:at select="//form-bean[1]/form-property[@name='amount']/@initial">
      <xr:override>100</xr:override>
   </xr:at>
   <xr:at select="/struts-config/form-beans/form-bean[1]">
      <xr:append>
         <form-property name="date" type="java.lang.String"/>
         <form-property name="dateFormat" type="java.lang.String"/>
      </xr:append>
   </xr:at>
</xr:refines>
```

Figure 3.9: First approach: A delta document that overrides an initial value of a `form-property` and adds new form-properties to a `form-bean`.

```
<struts-config xmlns:xr="http://www.onekin.org/XRefine"
               xr:artefact="struts-config.xml" xr:feature="dateRate_base">
   <form-beans>
      <form-bean name="ConverterForm"
                 type="org.apache.struts.action.DynaActionForm">
         <form-property name="amount" type="java.math.BigDecimal"
                        initial="100"/>
         <form-property name="sourceCurrency" type="java.lang.String"/>
         <form-property name="targetCurrency" type="java.lang.String"/>
         <form-property name="date" type="java.lang.String"/>
         <form-property name="dateFormat" type="java.lang.String"/>
      </form-bean>
   </form-beans>
   <!–content omitted –>
</struts-config>
```

Figure 3.10: The synthesised document from the composition of *dateRate* • *base*.

Let's consider that composition implies adding (i.e. appending (`<xr:append>`), prepending (`<xr:prepend>`) or overriding (`<xr:override>`)) the content of a node selected by an XPath expression, which is defined through the `<xr:at>` element[4]. In this way, *dateRate* can be supported as shown in Figure 3.9: the first `form-bean[1]` element is extended with the *date* and *dateFormat* properties whereas the value of the `initial` attribute is overridden from 1 to 100. Taken the artefact at figure 3.7(b) as a *base* document, a document exhibiting feature *dateRate* can be syn-

[4]The ***xr*** namespace prefix is used in this first approach.

thesised by composing *dateRate•base* (see figure 3.10[5]).

However, this implies handling XML documents as mere data structures where any element node can be subject to composition. In the same way that the `initial` attribute of a `<form-property>` element were changed, so it could have been the `type` attribute. However, this would have produced a runtime error due to it is not the expected type by the action. This is too fine-grained granularity that defeats the principle of modularity whereby high level abstractions (i.e. the modules) encapsulate their low level realisation (i.e. the instructions). Indeed, the *Open-Closed Principle* (OCP) [Mey97] states that modules "should be open for extension but closed for modifications" so that modules' behaviour can be extended without modifying their source code. This is especially valuable in a production environment, where changes to source code may necessitate code reviews, unit tests, and other such procedures to qualify it for use in a product: code obeying the principle doesn't change when it is extended, and therefore needs no such effort.

3.4.1.2 The second approach: schema-based modularization

This second attempt draws a distinction between elements playing the role of modules (and hence, being subject to composition) and elements that describe the realization of these modules (and hence, protected against external updates). Thus, an ***XML module*** is defined as an element of a document that carries out a specific function and it is liable to be re-used by/combined with other modules. Elements of a document are set by its schema. Schemas state the element, attribute and atomic type names, in addition to structural constraints that instances of this schema must obey. *XML Schema* is a W3C standard for vocabulary definition [TBMM01].

For our purposes, a way to indicate which elements play the role of XML modules is needed. For our sample case, we want to state that only element types `<struts-config>`, `<form-beans>`, `<form-bean>`, `<action-mappings>` and `<action>` can be modules (liable to be refined). The rest of the element types can not be refined (e.g. `<controller>` served only for implementation). To this end,

[5]The "`<!--content omitted-->`" comment is used to omit the content that is not relevant for the example.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xak="http://www.onekin.org/xak">
  <xs:element name="struts-config" xak:modularizable="yes"
              type="struts-configType"/>
  <xs:element name="form-beans" xak:modularizable="yes"
              type="form-beansType"/>
  <xs:element name="form-bean" xak:modularizable="yes"
              type="form-beanType"/>
  <xs:complexType name="form-beanType">
    <xs:sequence>
      <xs:element ref="description" minOccurs="0"/>
      <xs:element ref="form-property" minOccurs="0"
                  maxOccurs="unbounded"/>
    </xs:sequence>
    <!–content omitted –>
  </xs:complexType>
  <xs:element name="form-property" type="form-propertyType"/>
  <xs:element name="action-mappings" xak:modularizable="yes"
              type="action-mappingsType"/>
  <xs:element name="action" xak:modularizable="yes"
              type="actionType"/>
  <xs:element name="controller" type="controllerType"/>
  <!–content omitted –>
</xs:schema>
```

Figure 3.11: Schema-based modularisation.

the *xak:modularizable* attribute[6] is introduced. Figure 3.11 shows an excerpt of the XML Schema for the Struts' schema, now annotated with this attribute. The *xak:modularizable* attribute indicates whether an element type is eligible to be a module or not. For our example, the schema indicates that for instance, *<action>* and *<form-bean>* are modules, i.e. they can be refined, whereas *<controller>* is not amenable to refinement. Notice that for a given document instance, this does not force every occurrence of a modularizable element type (e.g. *<action>)* to be refined, but prevents non-modularizable element types (e.g. *<controller>*) from being refined.

Stating modularity at the schema level let the schema designer decide what elements play the role of a module. However, the schema just defines the expressiveness of the language (e.g. Struts) but not the semantics of the application

[6]The *xak* prefix denotes that this attribute belongs to the XAK namespace, i.e. *http://www.onekin.org/xak*.

being implemented. This is achieved by programmers. Programmers know the semantics of the application being defined, and hence, mechanisms should be available for them to indicate what elements make sense to be turned into modules. This is similar to the “final” attribute for Java classes. Java’s classes is the counterpart of modules. Classes can be refined. However, the language permits programmes to limit this option by using the “final” attribute. A similar mechanism is needed for XAK. Hence, *schema-based* modularisation is complemented with *instance-based* modularisation

3.4.1.3 The third approach: instance-based modularization

Schema-based modularization imposses necessary but not sufficient conditions. That is, the schema sets which element types are eligible for a module, but not all elements of this type need to be a visible module for composition in a document instance of this schema. The visible modules of a document will define the XAK document interface for composition.

Means are needed for defining which elements play the role of a module in instance documents. This is the role of the `xak:module` attribute. Back to the running example, consider that only *“/Page.Converter.events*” is a refinable `<action>;` whereas *“/Action.ConvertNow*” and those related to displays (e.g. *“/Page. Converter.display*”) can not be refined. This is stated as shown in figure 3.12. This moves the decision of what can be refined to the instance level.

Besides realizing abstractions, modules should be univocally identified. Xpath [CD99] could be used for this purpose by identifying the module through its location within the document. Unfortunately, Xpath expression are position dependent. If the position of the element changes, so does the Xpath expression. Therefore, location-based Xpath expressions can not be used for element identification when the position of this element is amenable to be changed, as it is the case for deltas. In the lack of dependencies that dictates something else, delta composition should be ideally commutative: $Feature1 \bullet Feature2 \bullet Base = Feature2 \bullet Feature1 \bullet Base$. This implies that the very same element can be located at different places. Hence, Xpath can not be used for module identification, and an ID is introduced for module addressing.

```
<struts-config xmlns:xak="http://www.onekin.org/xak"
               xak:artefact="struts-config.xml" xak:feature="base">
   <form-beans xak:module="mForms">
      <form-bean name="ConverterForm" xak:module="mConverterForm">
         <!–content omitted –>
      </form-bean>
   </form-beans>
   <action-mappings xak:module="mActions">
      <action path="/Page.Converter.display" .../>
      <!–content omitted –>
      <action path="/Page.Converter.events" xak:module="mEvents" ...>
         <!–content omitted –>
      </action>
      <action path="/Action.ConvertNow" ...>
         <!–content omitted –>
      </action>
   </action-mappings>
</struts-config>
```

Figure 3.12: Instance-based modularisation.

Schema-based and *instance-based* approaches to module definition offers a good balance between the controlled approach that offers the schema, and the freedom that programmers' activity requires. This is akin to the openness and subsidiary way of working that characterises the XML world (e.g. schema management in XML Schema), and that also exhibits SPLs [CN01]. SPLs introduce a main distinction between domain engineers, responsible for managing the core assets (i.e. the platform and the features), and application engineers, in charge of developing the product from the core assets. In this setting, *the platform designers* can use a schema-based approach to define the *"refinable"* element types, *the feature designers* can work at the instance level by indicating the concrete *"refinable"* elements, and finally, *the application designers* compose the features to synthesise the final application, refining some element contents, should it be required.

3.4.2 The ways of composition

A *class refinement* can introduce new data members, methods and constructors to a target class, as well as extend or override existing methods and constructors of

Jak	XAK
class	document
method	module
class refinement	delta document
method refinement	delta module

Figure 3.13: Same concepts, slightly different terminology in Jak and XAK.

that class [BSR04]. Likewise, *XML deltas* can add, extend or override existing module elements. Table3.13 shows the mapping between Jak terminology and XAK terminology. This requires: *(i)* a namespace to indicate which elements play the role of modules and their delta, and *(ii)* to realise the operational semantics of composition for XML modules by overloading the composition operator • [Bat].

3.4.2.1 The XAK namespace

Product synthesis starts with a base product and apply deltas to progressively incorporate new features to this product. Thus, there exists two kinds of XAK artefacts: *base documents* and *delta documents*.

Base document. Traditional XML documents can play the role of base XAK document. The difference stems from distinguishing between XML elements liable to be refined (i.e. modules), and those that can not be refined (i.e. the implementation). To this end, the XAK namespace provides three attributes (see figure 3.12), namely:

- `@xak:artefact`[7], which specifies the name of the document that is being incrementally defined;
- `@xak:feature`, which indicates the name of the feature being supported[8]; and
- `@xak:module`, which identifies those elements that play the role of modules.

[7] @ means attribute in XPath notation.
[8] For base documents, this attribute keeps the value *"base"*.

The latter keeps the module identifier whereas the element's content corresponds to the module implementation. Notice that the condition whereby module elements pertain to a modularizable type, is a necessary one, but it is not sufficient: the designer is not forced to turn into modules all elements of a modularizable type.

Delta documents. A delta is an increment in program's functionality. In addition to the previous *xak:module* attribute for delta module definition, a delta document is specified through two elements:

- *<xak:refines>* , which is the root element of the delta document
- *<xak:keep-content/>* , which dictates to inherit the content of the module being refined. It is similar to the "super" construct in some OO programming languages.

As an example of the former, consider the following snippet:

```
<xak:refines xmlns:xak="http://www.onekin.org/xak"
        xak:artefact="struts-config.xml" xak:feature="customisation">...
</xak:refines>
```

This introduces a delta document that supports the *"customisation"* feature. Its content describes a set of module refinements (delta modules, i.e. elements annotated with the *xak:module* attribute) over a given base document (i.e. the *xak:artefact* attribute). Moreover, the *@xak:feature* attribute specifies to which feature this delta belongs to so that all deltas pertaining to the same feature are unitedly applied.

An example for *<xak:keep-content/>* follows:

```
<action xak:module="mEvents" xak:keep-attributes="yes">
   <xak:keep-content/>
   <forward name="customise" path="/Page.Customise.display.do"/>
</action>
```

This snippet refines the *mEvents* module by adding a `<forward>` element to its previous content. The composer will replace the `<xak:keep-content/>` element (previous to composition) by the content of the module. Conceptually, this is a kind of cross-reference. But instead of pointing to its content, it transcludes it[9]. There is however an important difference in how transclusion is used in other context (e.g. entities in DTD [GP98] or *XInclude* [W3C02]). The semantic of the `<xak:keep-content/>`'s transclusion is restricted to the content of the module. This means that IDs and other attribute elements are outside of reusing. Hence, in the same way that the refinement of a class does not involve re-naming this class, the refinement of an element keeps unchanged the element's attributes. The element's attributes are viewed as the signature (i.e. metadata) of the module, whereas the element's content realises the implementation.

Unfortunately, the distinction between attributes and elements is not clear in the XML world [Ogb04], [OAS04]. Besides some syntactic differences, data can be supported as part of the element content (thus, being refinable) or as an attribute (thus, being no refinable). As schema designers are not aware of our convention, the `@xak:keep-attributes` attribute acts as a trapdoor for developers. It can hold two values: *(i)* "yes", permits to maintain all the attributes of the original module and only the redefined ones are override; *(ii)* "no", none of the attributes is maintained from the original module except the ID attribute. In this case, the developer is in charge of defining all the required attribute except the ID.

Back to the running example, let us consider the *customisation* feature. This feature provides the functionality for users to be able to set default values for the *sourceCurrency*, the *targetCurrency* and the *currencyList* properties. This feature impacts all, the model, the view and the controller. Hence, this feature is realized through three deltas whose jointly usage is ensured by all referring to the the same feature in their headings: `@xak:feature = "customisation"`. Specifically:

- model enhancement is achieved through *customiseAction.java*. This refinement adds a class for implementing the new action,
- view enhancement is accomplished through *customise.jsp*. This refinement adds a web page for implementing the interface of the new action

[9]Transclusion is the inclusion of part of a document into another document by reference [http://en.wikipedia.org/wiki/Transclusion].

```
<xak:refines xmlns:xak="http://www.onekin.org/xak"
             xak:artefact="struts-config.xml" xak:feature="customisation">
   <form-beans xak:module="mForms">
      <xak:keep-content/>
      <form-bean name="CustomiseForm" xak:module="mCustomiseForm"
                 type="org.struts.actions.DynaActionForm">
         <form-property name="defaultSourceCurrency" type="java.lang.String"/>
         <form-property name="defaultTargetCurrency" type="java.lang.String"/>
         <form-property name="currencyList" type="java.lang.String[]"/>
         <form-property name="menuRows" initial="7" type="java.lang.Integer"/>
      </form-bean>
   </form-beans>
   <action xak:module="mEvents" xak:keep-attributes="yes">
      <xak:keep-content/>
      <forward name="customise" path="/Page.Customise.display.do"/>
   </action>
   <action-mappings xak:module="mActions">
      <xak:keep-content/>
      <action path="/Page.Customise.display" forward="/customise.jsp"/>
      <action path="/Action.Customise" name="CustomiseForm" scope="session"
              type="org.onekin.xak.example.struts.actions.CustomiseAction">
         <forward name="hasSuccedded" path="/Page.Converter.display.do"/>
         <forward name="hasFailed" path="/Page.Error.display.do"/>
      </action>
   </action-mappings>
</xak:refines>
```

Figure 3.14: The *customisation* XAK delta document.

- controller enhancement is accomplished through *struts-config.xml* (see Figure 3.14). This refinement *(i)* adds the *CustomiseForm* form-bean definition into the existing mForms module; *(ii)* extends the *mEvents* dispatcher action to show the *Customise* page of the feature, and *(iii)* defines the control-flow related to the new action.

Consider

- the *base* XAK document in Figure 3.12, where *mForms*, *mActions* and *mEvents* are set as modules,
- the *customisation* XAK delta that is shown in Figure 3.14.

Then, *customisation* • *base* will deliver the enhanced *strust-config.xml* document shown in figure 3.15. Notice that the composition ripples through all artefacts whose headings include `@xak:feature = “customisation”`.

```
<struts-config xmlns:xak="http://www.onekin.org/xak"
               xak:artefact="struts-config.xml" xak:feature="customisation_base">
   <form-beans xak:module="mForms">
      <form-bean name="ConverterForm" xak:module="mConverterForm" ...>
         <!–content omitted –>
      </form-bean>
      <form-bean name="CustomiseForm" xak:module="mCustomiseForm"
                     type="org.struts.actions.DynaActionForm">
         <!–content omitted –>
      </form-bean>
   </form-beans>
   <action-mappings xak:module="mActions">
      <action path="/Page.Converter.events" name="ConverterForm"
               xak:module="mEvents"
               type="org.onekin.xak.example.struts.actions.ConvertNowAction">
         <forward name="convertNow" path="/Action.ConvertNow.do"/>
         <forward name="cheatsheet" path="/Action.Cheatsheet.do"/>
         <forward name="customise" path="/Page.Customise.display.do"/>
      </action>
      <action path="/Action.ConvertNow" ...>
         <!–content omitted –>
      </action>
      <!–content omitted –>
      <action path="/Page.Customise.display" forward="/customise.jsp"/>
      <action path="/Action.Customise" name="CustomiseForm" scope="session"
               type="org.onekin.xak.example.struts.actions.CustomiseAction">
         <forward name="hasSuccedded" path="/Page.Converter.display.do"/>
         <forward name="hasFailed" path="/Page.Error.display.do"/>
      </action>
   </action-mappings>
</struts-config>
```

Figure 3.15: The synthesised document from the composition *customisation • base*.

3.4.3 The laws of composition

SPLs last longer and involve larger teams than those of single products. Specifically, refinements (i.e. feature realisations) might well be defined separately for different engineers at different times along distinct release agendas. Therefore, the information available at refinement-definition time basically includes the Feature Model and the *base*. Domain engineers should make minimal assumption about how other features are realised, except those explicitly capture in the Feature Model through feature dependencies. Nevertheless, refinements should smoothly fit together when synthesising to derive SPL products. While it is possible to check individual products by building and then compiling them, this is impractical. In a SPL, there can be thousands of products; it is more desirable to ensure that all legal realizations are synthesize-safe without enumerating the entire product line and compiling each product. In other words, mismatches at product-synthesize time should be minimised. In this setting, we rise the following question:

> **which sort of verification can be conducted at refinement-definition time to reduce the number of potential mismatches at product-synthetized time?**

Three kinds of errors can occur in a feature implementation [KAT+09]:

- *syntactic errors*. Syntax errors occur when a delta is ill-formed regarding the language's syntax, for example when an opened XML tag is not closed. These kind of errors are the easiest and are detected by an XML parser.
- *type errors*. Type errors occur when the delta is ill-formed regarding the language's type system, e.g., a non-declared XML element in XML-Schema is used to define a delta. These kind of errors are detected by validating a document against to XML-Schema.
- *semantic errors*. Semantic errors occur when the delta behaves incorrectly according to some (formal or informal) specification. They are the most difficult to detect.

This section focuses on type errors. Main errors at refinement-definition time include:

- the root element of the document does not identify the name of the feature or the document that implements (i.e. it does not contain `@xak:feature` or `@xak:artefact`),
- a `@xak:module` attribute is used in an element that is not modularizable according to the XML-Schema,
- the content of a base module is not a valid fragment (see later),
- the content of a delta module is not a valid delta fragment (see later).

Let S be a schema. A fragment is the content of a XAK module that is amenable to be turned into a valid module w.r.t S through composition. That is, F1 is a fragment if there exists another document D so that *"F1 • D"* yields a valid document. Notice that S-compliant modules (or documents) trivially satisfied this condition, being D the empty module (or document). The challenge is when *F1* is not S-compliant. In this case, *F1* can still be a fragment if it itself does not compromise the rules set by S. For example, suppose that a module type defines the `<a/><b/><c/>` sequence as its content model. Then, the valid fragment set include: `<a/>` ; `<b/>` ; `<c/>` ; `<a/><b/>` ; `<b/><c/>` ; `<a/><b/><c/>`. Notice that `<a/><c/>` is not a fragment since it is impossible to generate a valid module (or document) no matter who it is composited with.

Let S be a schema. A delta fragment is the content of a delta module. A delta fragment is defined by combining the `<xak:keep-content/>` element with valid fragments. Some examples follow:

- `<xak:keep-content/><a/><c/>` is not a valid delta fragment because `<a/><c/>` is not a fragment,
- `<xak:keep-content/><a/>` is not a valid delta, too. Here, `<a/>` is a fragment but `<xak:keep-content/>` can not hold any content that makes the composition valid w.r.t. `<a/><b/><c/>`
- `<xak:keep-content/><b/><c/>` is a valid delta, where `<xak:keep-content/>` can be substituted by `<a/>` to yield a valid document
- `<b/><xak:keep-content/><c/>` is not a valid delta, since `<xak:keep-content/>` can not be substituted by any fragment that makes the result valid, i.e. S-compliant
- `<a/><xak:keep-content/><c/>` is a delta where `<xak:keep-content/>` can be substituted by `<b/>` to yield a valid document
- `<a/><xak:keep-content/>` *is a delta* where `<xak:keep-content/>` can be substituted by `<b/> or <b/><c/>` to yield a valid document.

Summarizing, a delta fragment type should define the following rules:

- at least a S-compliant fragment is defined,
- in case that `<xak:keep-content/>` is used:
  - it can be used only once for maintaining the previous content, (i.e. a S-compliant fragment)
  - the added S-compliant fragments are restricted to be placed at the edges of the module's content (i.e. at the beginning or at the end).
  - the previous two rules should be combined without violating the preceding order of sequential elements in a S-compliant fragment type. Thus, the composer will yield a valid S-compliant fragment after composition;

Feature realisations are deltas. We want deltas to be validated at the time they are defined without waiting for the product to be derived. In the XML world, validation is achieved through schemas. The question arises about:

> *how to obtain delta schemas (e.g.$\triangle S$) from document schemas (e.g. S)?*

Definition of $\triangle S$ has a main stumbling block. Deltas are compound documents. Compound Document is the W3C term for a document that combines vocabularies. Deltas are Compound Documents in so far as they intermingle elements from two namespaces: the *XAK* and the *S* namespaces. The challenge is to come up with a (compound) schema so that *standard* XML validators can be used for delta validation.

This issue also arises in other settings, for instance, the combined use of XHTML and XForms in the same document. XForms can use XHTML elements (e.g. *<xhtml:div>*), and XHTML can use XForms elements (e.g. *<xforms:repeat>*). This permits the existence of an *<xhtml:div>* element which contains an *<xforms:repeat>* which, in turns, keeps an *<xhtml:div>*, and so on. Current validators overlook this situation and validate as correct multiple nesting of any XForms and XHTML tag (e.g. if *<xsd:any>* is used in the schema definition of each namespace). However, an editor that supports multi-namespace documents or a schema validator that strives to produce accurate error messages, must have specific information about how tags from two distinct namespaces can be correctly intermingled.

Combining schemas can be problematic: elements from schema A can not always be freely intertwined with elements from schema B, i.e. the content model of A's elements can be violated when embedding elements of schema B (e.g. `<xform:form>` of XForms do not allow `<xhtml:table>` from XHTML [SK05a, SK05b]). These restrictions of incompatibilities among the content models of elements coming from different schemas are reflected through the so-called *compound document profiles* [W3C04].

Profiles define how elements from different namespaces can be intermingled. Profiles exist for XHTML+XForms, XHTML/X+V (a subset of VoiceXML), and a specific compound XML document editor exists for Eclipse [KKW05]. Realised through an schema, a profile introduces additional restrictions on how elements from different namespaces can be combined. Usually, these elements represent reusable components outside of its namespace. A profile redefines content models of elements in order to define the new constraints. A profile is then a grammar, too. Hence, the term *compound grammars* is also used as a synonyms for profiles The big deal is that profiles permit to validate compound documents using standard validators while increasing the detection and accuracy of errors at validation (rather than processing) time. This is our approach: *delta grammars* ($\triangle S$).

3.5 Artefact Design: Delta Grammars

Previous section has intuitively introduced the notion of delta grammars. This section provides a systematic description of how delta grammars can be obtained. First, a background of regular expressions is introduced for representing formally XML types defined in an XML Schema. Second, the validation problem is presented as a type-checking problem. Finally, we explain the algorithms to obtain the delta grammars.

3.5.1 Regular expression types

We want to statically validate that deltas are valid. This requires of regular expression types to describe XML types of the XML Schema at hand. To this end, we resort to the regular expression notation introduced in XDuce [HVP05]. We need to define *types*, which in turn, requires of *labels* and *values* for validation purposes.

Labels. Assume that the names of elements, types and attributes defined in a given schema are described by a (infinite) set of labels, range over by *l*.

$$\begin{array}{rcl} L ::= & l & label \\ & any & wildcard\,label \\ & L|L & union \\ & L\backslash L & difference \end{array}$$

Values. For brevity, we omit base values such as strings. (The changes required to add them are straightforward.) We assume a countably infinite set of labels, ranged over by *l*. Values are defined as follows:

$$v \quad ::= \quad l_1[v],..,l_n[v] \qquad (n \geq 0).$$

We write () for the empty sequence and v, w for the concatenation of sequences v and w.

Types. *Syntax*. We assume given a countably infinite set of type names, ranged over by X. Type expressions are now defined as follows:

$$\begin{array}{rcl} T ::= & () & empty\,sequence \\ & X & type\,variable \\ & l[T] & label \\ & T\,|\,T & union \\ & T,T & concatenation \\ & T+ & repetition \\ & \emptyset & empty\,set \end{array}$$

The regular expression operators *, and ? are obtained as syntactic sugar:

$T? \equiv T \mid ()$

$T* \equiv T + |()$

The interpretation of type names are given by a single, global set E of type definitions of the following form:

$type X = T$

The body of each definition may mention any of the defined variables (in particular, definitions may be recursive). Consider $E : X \rightarrow T$ as a mapping function from type names to their bodies and write *E(X)* for the right-hand side of the definition of *X* in *E*.

To ensure that types correspond to regular tree automata (rather than context-free grammars), we impose a syntactic restriction that disallows recursion "at the top level" of definitions. For a given type *T*, we define the set S(T) of type names reachable from *T* at the top level as the smallest set satisfying the following:

$$S(T) \quad = \quad \begin{cases} S(E(X)) \cup \{X\} & if\,T = X \\ S(T_1) & if\,T = T_1+ \\ S(T_1) \cup S(T_2) & if\,T = T_1, T_2\,or\,T = T_1 \,|\, T_2 \\ \emptyset & otherwise \end{cases}$$

We then require that the set E of type definitions satisfies:

$$X \notin S(E(X))\;for\,all\,X \in dom(E).$$

type Tstruts-config = ***struts-config****[Tform-beans, Taction-mappings]*
type Tform-beans = ***form-beans****[Tform-bean*]*
type Tform-bean = ***form-bean****[description[] ?, form-property[] *]*
type Taction-mappings = ***action-mappings****[Taction*]*
type Taction = ***action****[forward[]*]*
...

Figure 3.16: An excerpt of the *struts-config* type using *XDuce* notation.

The semantic of types is given by the relation $v \in T$, read "value v has type T". This mean that v has no type errors.

3.5.2 The validation of XML documents

An XML document is essentially an ordered labeled tree. For the purpose of type checking, we ignore data values and only consider their atomic types. Thus, we fix an alphabet L of tag names, attribute names (those with the @ prefix), and atomic type names. For a simple illustration, the alphabet of the document in figure 3.7 is:

$$\begin{aligned} L &= \{struts-config, form-beans, form-bean, @name, @type, \\ &\quad form-property, @initial, action-mappings, action, xs:string, \ldots\} \end{aligned}$$

Let $\mathcal{T}_\mathrm{L}$ denote *a regular tree language* over alphabet L, i.e. the set of ordered trees where each node is labeled with an element from L**.** An XML Schema S is a subset of $\mathcal{T}_\mathrm{L}$ and defines an XML type τ. This type τ is in turn defined by a set of type identifiers, X, and associates to each identifier a regular expression over LxT.

> **Definition**. *Given a document $d \in \mathcal{T}_\mathrm{L}$ and a type $\tau \subseteq \mathcal{T}_\mathrm{L}$ (defined over an XML Schema S), validation refers to the process of deciding whether $d \in \tau$, in which case d is said to be valid with respect to S or d is S-compliant[Suc02].*

Figure 3.16 shows an excerpt of an XML type, which describes the *Struts-config* schema (i.e. the *struts-config* type) using XDuce notation [HVP05]. The type *Tstruts-config* stands for a regular expression where the element *struts-config* exhibits a sequential structure whose nodes go along the *Tform-beans* and *Taction-*

Table 3.1: Distinct (input,output) scenarios for the function $fragmentT : T \longrightarrow T$

Input (a type)	**Output (a fragment type)**	informative
	(deterministic grammar)	(non-deterministic grammar)
$()$	$\emptyset$	
$a[T_1]$	$a[T_1]$	
$a[T_1] \mid b[T_2]$	$a[T_1] \mid b[T_2]$	
$a[T_1], b[T_2]$	$a[T_1], X_b? \mid X_b \Rightarrow$ $\{\vdash type X_b = b[T_2]\}$	$\equiv a[T_1] \mid b[T_2] \mid a[T_1], b[T_2]$
$a[T_1], b[T_2]*$	$a[T_1], X_b? \mid X_b \Rightarrow$ $\{\vdash type X_b = b[T_2]+\}$	$\equiv a[T_1] \mid b[T_2]+ \mid$ $a[T_1], b[T_2]+$
$a[T_1]*, b[T_2], c[T_3]+$	$a[T_1]+, X_b? \mid X_b \Rightarrow$ $\{\vdash type X_b = b[T_2], X_c? \mid X_c;$ $type X_c = c[T_3]+\}$	$\equiv a[T_1]+ \mid b[T_2] \mid c[T_3]+ \mid$ $(a[T_1]+, b[T_2]) \mid$ $(b[T_2], c[T_3]+) \mid$ $a[T_1]*, b[T_2], c[T_3]+$

mappings types. This basically defines the content model[10] of the *struts-config* element. In this work, the type of a sequence of attributes is not declared in order to simplify the example. For example,figure 3.7b) shows a *valid* document with respect to the *Tstruts-config* type (defined over the *Struts-config* schema).

3.5.3 Obtaining the delta grammars: △*fragmentType*

This subsection introduces an algorithm to obtain delta module and fragment grammars. The algorithm takes as input an schema S (e.g. Struts), and returns another schema $\triangle S$ for validating deltas (i.e. Struts-compliant). This algorithm rests on the existence of the following functions: $fragmentT, firstKc, insertKc,$ $\triangle fragmentT, \triangle moduleT\ and\ generate - delta - grammar$. Next paragraphs describe these functions in reverse order of usage. The XML schema describes a set of XML types. In this work, XML types are represented by regular expressions using XDuce syntax.

• $fragmentT : T \longrightarrow T$. This function takes a type T, and delivers a fragment type that defines all the possible fragments of T. A fragment type does not include the empty sequence. Moreover, a set of types related to the built type expression are added to the system for optimizing it.

[10]For simplification sake, attribute types are removed from the example.

$$
\begin{aligned}
fragmentT(\emptyset) &= \emptyset && (3.1)\\
fragmentT(()) &= \emptyset && (3.2)\\
fragmentT(X) &= fragmentT(E(X)) && (3.3)\\
fragmentT(l[T]) &= l[T] && (3.4)\\
fragmentT(T_1 \,|\, T_2) &= fragmentT(T_1) \,|\, fragmentT(T_2) && (3.5)\\
fragmentT(l[T]+) &= l[T]+ && (3.6)\\
fragmentT(\emptyset, T) &= \emptyset && (3.7)\\
fragmentT((), T) &= fragmentT(T) && (3.8)\\
fragmentT(X, T) &= fragmentT(E(X)) && (3.9)\\
fragmentT(l[T_1], T_2) &= l[T_1], X_{T_2}? \,|\, X_{T_2}) \Rightarrow && (3.10)\\
 & \{\vdash type\, X_{T_2} = fragmenT(T_2)\} && (3.11)\\
fragmentT(l[T_1]+, T_2) &= l[T_1]+, X_{T_2}? \,|\, X_{T_2}) \Rightarrow && (3.12)\\
 & \{\vdash type\, X_{T_2} = fragmenT(T_2)\} && (3.13)\\
fragmentT((T_1 \,|\, T_2), T_3) &= fragmentT(T_1, T_3) \,|\, fragmentT(T_2, T_3) && (3.14)\\
fragmentT((T_1, T_2), T_3) &= fragmentT(T_1, (T_2, T_3)) && (3.15)
\end{aligned}
$$

These rules generalize the example we have used in subsection 3.4.3. This function at work is shown in table 3.1. For example, if the given type expression is $a[T_1], b[T_2]*$, then the $a[T_1], X_b? \,|\, X_b$ type expression is obtained. First, the given type expression is expanded replacing the * operator, $a[T_1], (b[T_2] + \,|\, ())$. After that, the 3.10 rule is applied, and then follows by 3.5 and 3.6. Also, the empty ( ) sequence is removed by 3.2, since it is not permitted to use as a fragment. During this process, the $\{\vdash type\, X_b = b[T_2]+\}$ has been added to the list of defined types in order to optimize the derived regular expression. The purpose of third column of table 3.1 is informative for watching all the alternatives and notice the expasion that produce the fragment type definition.

• $firstKc : T \longrightarrow T$. This function takes as input type T, and delivers a new type that permits to use `<xak:keep-content/>` as first element. This element follows with a fragment type of the given type. Moreover, a set of types related to the built type expression are added to the system for optimizing it.

$$
\begin{array}{rcl}
firstKC(T) & = & xak:keep-content[], X_T \Rightarrow \\
 & & \{\vdash typeX_T = fragmenT(T_2)\}
\end{array}
$$

• $insertKc : T \longrightarrow T$ This function takes as input a type T, and delivers a new type that keeps the compliant fragments of the given type and also combines the *<xak:keep-content/>* element with possible fragments of the given type T. The possible fragments before the *<xak:keep-content/>* element are generated step-by-step for deriving a deterministic grammar, whereas the possible fragments after the `<xak:keep-content/>` element are generated using the above *fragmentT*. Moreover, a set of types related to the built type expression are added to the system for optimizing it.

$$
\begin{array}{rcl}
insertKC(\emptyset) & = & \emptyset \\
insertKC(()) & = & \emptyset \\
insertKC(l[T]) & = & l[T] \\
insertKC(X) & = & insertKC(E(X)) \\
insertKC(T_1 \,|\, T_2) & = & insertKC(T_1) \,|\, insertKC(T_2) \\
insertKC(l[T]+) & = & l[T_1]+, (xak:keep-content[], X_l?)? \\
 & & \{\vdash typeX_l = l[T_1]+\} \\
insertKC(\emptyset, T) & = & \emptyset \\
insertKC((), T) & = & insertKC(T) \\
insertKC(X, T) & = & insertKC(E(X), T) \\
insertKC(l[T_1], T_2) & = & l[T_1], (xak:keep-content[], X_{T_2}? \,|\, X_{T_2-kc})? \,|\, X_{T_2-kc} \Rightarrow \\
 & & \{\vdash typeX_{T_2} = fragmenT(T_2); \\
 & & typeX_{T_2-kc} = insertKC(T_2)\} \\
insertKC(l[T_1]+, T_2) & = & l[T_1]+, (xak:keep-content[], X_l? \,|\, X_{T_2-kc})? \,|\, X_{T_2-kc} \Rightarrow \\
 & & \{\vdash typeX_l = fragmenT(l[T_1]+, T_2); \\
 & & typeX_{T_2-kc} = insertKC(T_2)\} \\
insertKC((T_1 \,|\, T_2), T_3) & = & insertKC(T_1, T_3) \,|\, insertKC(T_2, T_3) \\
insertKC((T_1, T_2), T_3) & = & insertKC(T_1, (T_2, T_3))
\end{array}
$$

• $\triangle fragmentT : T \longrightarrow T$. This function takes the content model of a *moduleType*, and delivers a T-compliant $\triangle$*fragmentType.* It includes the *fragmentTypes* of the given type and how they can be combined with the `<xak:keep-content/>` element. Moreover, a set of types related to the built type expression are added to the system for optimizing it.

$$\begin{aligned}
\triangle fragmentT(\emptyset) &= \emptyset \\
\triangle fragmentT(()) &= \emptyset \\
\triangle fragmentT(X) &= \triangle fragmentT(E(X)) \\
\triangle fragmentT(l[T]) &= l[T] && (3.16) \\
\triangle fragmentT(T_1 \,|\, T_2) &= \triangle fragmentT(T_1) \,|\, \triangle fragmentT(T_2) \\
\triangle fragmentT(l[T]+) &= insertKC(l[T_1]+) \,|\, firstKC(l[T_1]+) && (3.17) \\
\triangle fragmentT(\emptyset, T) &= \emptyset \\
\triangle fragmentT((), T) &= \triangle fragmentT(T) \\
\triangle fragmentT(X, T) &= \triangle fragmentT(E(X), T) \\
\triangle fragmentT(l[T_1], T_2) &= insertKC(l[T_1], T_2) \,|\, firstKC(T_2) && (3.18) \\
\triangle fragmentT(l[T_1]+, T_2) &= insertKC(l[T_1]+, T_2) \,|\, firstKC((l[T_1]+, T_2)) \\
\triangle fragmentT((T_1 \,|\, T_2), T_3) &= \triangle fragmentT(T_1, T_3) \,|\, \triangle fragmentT(T_2, T_3) \\
\triangle fragmentT((T_1, T_2), T_3) &= \triangle fragmentT(T_1, (T_2, T_3))
\end{aligned}$$

These rules generalize the example we have used in subsection 3.4.3. The algorithm derives a delta fragment type that defines two kind of sequences: 1) fragments compliant to the given type and 2) fragments that refine an existing fragment. The latter uses the `<xak:keep-content/>` element and requires to add a fragment either before or after the existing one.

Mainly, this inclusion alters both the sequencing and cardinality of the sequential elements of the given type (i.e. the content model of a module type declared in S). Grammar complexity mainly stems from the different placements of the `<xak:keep-content/>` within the original content of the module. This placement depends on the cardinalities of the sequential elements in the given type.

Some (input, output) scenarios for this function are given in table 3.2. The introduction of `<xak:keep-content/>` considerably enlarges the content alternatives:

- **no extension permitted** (first three rows). It stands for the **overriding** case.

Table 3.2: Distinct (input,output) scenarios for the function $\triangle fragmentT : T \longrightarrow T$

Input (a type T)	**Output (a delta fragment type △fT)**
()	$\emptyset$
$a[T_1]$	$a[T_1]$
$a[T_1] \mid b[T_2]$	$a[T_1] \mid b[T_2]$
$a[T_1], b[T_2]$	$a[T_1], xak:keep-content[]? \mid X_{b-kc} \mid$ $xak:keep-content[], X_b \Rightarrow$ $\{\vdash type X_{b-kc} = b[T_2]\, ; type X_b = b[T_2]\}$
$a[T_1], b[T_2]*$	$a[T_1], (xak:keep-content[], X_b? \mid X_{b-kc})? \mid X_{b-kc} \mid$ $xak:keep-content[], X_b \Rightarrow$ $\{\vdash type X_b = b[T_2]+;$ $type X_{b-kc} = b[T_2]+, xak:keep-content[]?, X_b?\}$
$a[T_1]+$	$a[T_1]+, (xak:keep-content[], X_a? \mid$ $xak:keep-content[], X_a \Rightarrow$ $\{\vdash type X_a = a[T_1]+\}$
$a[T_1]*, b[T_2], c[T_3]+$	$a[T_1]+, (xak:keep-content[], X_a? \mid X_{b-kc})? \mid X_{b-kc} \mid$ $xak:keep-content[], X_a) \Rightarrow$ $\{\vdash type X_a = a[T_1]+, X_b? \mid X_b\, ;$ $type X_b = b[T_2], X_c? \mid X_c\, ;$ $type X_c = c[T_3]+;$ $type X_{b-kc} = b[T_2], (xak:keep-content[], X_c? \mid X_{c-kc})? \mid$ $X_{c-kc}\, ;$ $type X_{c-kc} = c[T_3]+, xak:keep-content[]?, X_c?\}$

The delta module should already contain a valid fragment of the given type.

- **no extension permitted for some fragments after** `<xak:keep-content/>` (four and five row). If a fragment starts with an element such that (1) it appears in the first position, and (2) its maxOccurs cardinality is 1 (e.g. $a[T_1]$), then it is not permitted to use it after `<xak:keep-content/>`. This is due to `<xak:keep-content/>` can not be substituted by fragments that make the result valid, i.e. S-compliant. These combinations are restricted by 3.16 and 3.18 rules.

- **no extension permitted for some fragments before** `<xak:keep-content/>` (four row). If a fragment ends with an element such that (1) it appears in the last position, and (2) its maxOccurs cardinality is 1 (e.g. $b[T_2]$), then it is not permitted to use it before `<xak:keep-content/>`.

- **total freedom** (last two row). Those elements whose maxOccurs cardinality is >1 (i.e. + or *) permit to combine with `<xak:keep-content/>` element in any position: before, after or both. For example, if the given type is $a[T_1]+$, then the obtained $\triangle$*fragmentType* (from 3.17) defines the following delta fragments as valid:

  - *`<a/><a/><xak:keep-content/>`*, *before*
  - *`<xak:keep-content/><a/><a/>`*, *after*
  - *`<a/><a/><xak:keep-content/><a/><a/><a/>`*, *both*

• $\triangle moduleT : T \longrightarrow T$. This function takes the *moduleType* definition and transforms into T-compliant $\triangle$*moduleType* by adding the *@xak:module* attribute and modifying its content model with the $\triangle fT$ type obtained from the $\triangle$*fragmentT* function.

$$\begin{aligned} \triangle moduleT(l[T]) &= l[(@xak:module[xs:string], \triangle fT)] \Rightarrow \\ &\quad \{\vdash type \triangle fT = \triangle fragmentT(T)\} \end{aligned}$$

• $generate-delta-grammar : \overline{T} \longrightarrow \overline{T}$. This function takes a *schema S* and derives a *delta grammar* $\triangle S$. Broadly, this function transforms all *moduleTypes* in S into $\triangle$*moduleTypes* using $\triangle moduleT$, $top-level$ and *isModuleType* functions.

```
for all $T_i$ in top-level(S)                    $i \in 1..n$
    if isModuleType($T_i$) then
        $type\triangle mT_i = \triangle moduleT(E(T_i))$
else
        $T_i$
```

The $top-level$ function returns the set of types and element types that are defined at top-level in the XML-Schema. The *isModuleType* function checks if the given type is a module type definition, returning *True* if an element type definition is annotated as *`@xak:modularizable="yes"`*, and *False*, otherwise. Appendix C describes how the obtained delta grammar is represented through an XML-Schema using as an example an excerpt of the Struts XML-Schema.

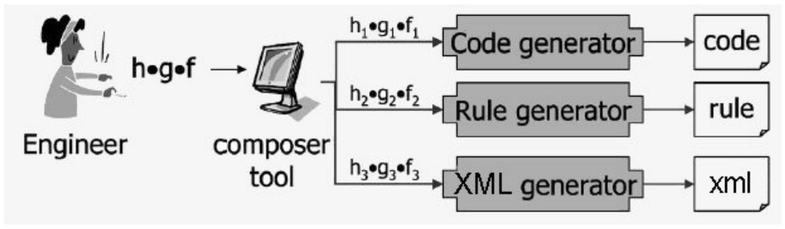


Figure 3.17: Organisation of AHEAD generators

3.6 Artefact Implementation: integrating XAK into AHEAD TS

This section describes the XAK composer and how they are integrated into AHEAD Tool Suite..

Instead of building one huge generator that deals with all possible program representations (which itself is impractical), AHEAD builds an elementary tool (i.e. *composer*) that expands a high-level equation into its constituent artefact equations (see Figure 3.17). Thus, a simple composer tool does the work of orchestrating other relatively simple artefact-type-specific tools to produce the complex set of artefacts that comprise a synthesised system[BSR04]. In this way, we define the *XAK composer* (i.e. XML generator), which is an artefact-type-specific composition operator for XML documents. Moreover, an Ant build task and script[Thea] are defined for integrating XAK into AHEAD.

From a composition perspective, base documents behave as values whereas delta documents behave as functions. The composer synthesises a new document by applying deltas on a base artefact (i.e. *-c* option). For instance, the enactment of *"customisation • base"* triggers the execution of the next command, which outputs the document shown in figure 3.15 (i.e. *-o* option):

```
> composer -c base/struts-config.xml customisation/struts-config.xml
          -o prog1/struts-config.xml
```

The bottom line is that XAK modules can be refined in a similar way to methods through class inheritance, i.e. by overriding the original module. If the purpose is to extend rather than substitute the original content, then the `<xak:keep-content/>`

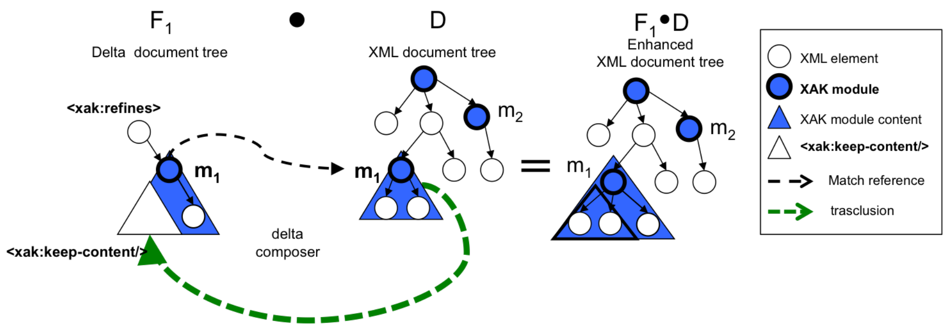


Figure 3.18: XAK module composition match modules with the same identifier

element indicates the point where to place the extended content, i.e. before or after this element. Therefore, XAK composer implements this XML generic composition over DOM tree representations. Figure 3.18 shows this situation. Let D, F_1 and m_1 be a base document, a delta and a XAK module. When realizing $F_1 \bullet D$ equation, the delta composer matches the namesake module and overrides the module in D with the new content in F_1. Previously, `<xak:keep-content/>` has been replaced by the content of the module in$D$. Thus, the delta composer yields a new enhanced document.

3.7 Artefact Evaluation

Both XAK namespace and composer has been used in seven scenarios:

1. *documenting SPL using DITA*[DAK09]. This paper presents a case study in a feature-oriented approach to SPL documentation where features are the guiding principle to conceive, organise and write documentation. To this end, we integrate IBM's DITA documentation architecture into SPL practices, realised by the AHEAD Tool Suite.

2. *a database reporting SPL* [ADT05]. This paper looks at database reports as a family of products within a SPL (i.e. reports are synthsized from a product-line) based on the predictability, similarity and variability among reports. Our test case is used to illustrate a product-line approach in XML-centric applications. A challenge is that data warehouse technology is not

available.

3. a simple *WebCalc web application*, and

4. a *production process for WebCalc* both presented at [DTA05]. This work describes how to apply variability to the synthesis process (described by Apache Ant scripts, i.e. *build.xml*). To attain this, it separates features that impact on the program and features that impact on how the program is built (i.e., the synthesis itself has features that change how it is done).

5. a *FlightReservation Portlet* [TBD07]. This work presents a case study on a product line of portlets, which are components of web portals. It shows how products in a software product line can be synthesized in an Model Driven Development (MDD) way by composing features to create models, and then transforming these models into executables. A model is defined by a specific DSL, which is represented by several XML documents.

6. an *SVG map* of US statistics. This web application is built folllowing a feature oriented paradigm, where the main core assets are SVG XML documents and Javascript files.

7. the largest is *the refactoring of ATS* [TBD06]. Trujillo et al. present a case study in feature refactoring a multi-representation program into a product line. The refactored program is the AHEAD Tool Suite (ATS), which is a collection of tools that were developed for feature-based program synthesis [Bat04]. ATS has been refactored into a *core* (the kernel of ATS) and optional features, one per tool (e.g., aj, cpp, drc, jedi, etc.). In addition to code, there are makefiles, regression tests, documentation, and program specifications, all of which are intimately intertwined into an integrated whole. After that, an essential tool in synthesizing variants of ATS is XAK.

Summarizing, the range of the generated products varies from a few LOCs (in the first) to 200 KLOCs (in the last). The percentage of XML files over total is around 10% (for ATS refactoring) while in web/Portlet applications are over 60 %.

XAK has been used to refine a variety of XML vocabularies such as: Apache Ant scripts, Struts configurations, XHTML pages, JSP pages, Portlet deployment descriptor, Web deployment descriptor, XCube data [HBH03], SVG graphics, XSL templates, SCXML statecharts [Bea06a], and xADL (Architecture Description Languages) [DVDHT05].

3.8 Related work

SPL programming. Among different programming paradigms to realize product families three can be highlighted in SPL: *Frame Technology* (FT) [Bas97], *Feature-Oriented Programming* (FOP) [Pre97]and *Aspect-Oriented Programming*(AOP) [ea97]. *XML-based Variant Configuration Language* (XVCL) [SZJ02], *Variability Specification Language* (VSL) [Bec02], AHEAD [BSR04], FeatureHouse [AKL09] are tools that follows the above paradigms in order to realize the variability in source code. XVCL and VSL are two approaches that are based on FT and use XML only for managing the variability issues (i.e. variation points and their variants). They can apply to XML documents, however they can have problems since they don't neither use namespace to separate distinct vocabularies nor an XML Schema in order to validate the variants. *AspectXML* and *HyperAdapt* [NKAM09] are two approaches that formalise the cross-cutting concerns of AOP for XML instances and schema. On one hand, AspectXML provides a clearer and hopefully easier means of specifying AOP without requiring that the developer understand the more complex superset of XSLT. The *AspectXML Engine* is implemented using plain XSLT and Xpath. As Xpath is fundamental to the pointcut definition in AspectXML, so is it the module identity in XAK. On the other hand, HyperAdapt defines which are the possible pointcut XML elements in the XML Schema and proposes the idea of static analysis based on type information derived from XML schema. These ideas are similar to our approach of *schema-based* modularisation and type-safe composition. Finally, The *XML FST-Composer* [Dör09] tool of FeatureHouse TS proposes a two-level modularisation (i.e. schema- and instance-based) by annotations, similar to our approach. He also presents the difficulties that has the XML technology for both modularisation and composition of documents, and implements a tool that extends the FSTComposer in order to integrate XML composition in FeatureHouse. FeatureHouse models software components by feature structure trees (FSTs) and FSTComposer com-poses software components represented by FSTs applying tree superimposition.

XML programming languages. Two main techniques to modify or create documents can be identified in XML: update operations and transformation languages. Examples of the former include *XUpdate* [LM00]and *XMLTask* [OOP]. Those approaches (used in different contexts) describe operations (represented in a declarative way) that can be performed over a tree representation of an XML document (i.e. DOM [Wea98]). They use an XPath expression to locate nodes in a given XML document, usually a single unique node is located (e.g. an element, attribute or text). Once the node which pinpoints the target for the modifications has been found, modifications like additions, insertions, removals or substitutions of elements and attributes can be done. They offer a total freedom of updating. However, the use of the XPath and the set of those low-level operations violate the rules of the unit of composition (i.e., identification and encapsulation) discussed in 3.4.1. Moreover, they haven't got any mechanism for statically type-checking the new XML fragment before updating.

In a similar way, XML transformation languages aim at facilitating the development of XML processing applications. Examples include XML-based and declarative languages, such as XSLT [Cla99] and XQuery [Bea06b], XML-centric languages, such as XDuce [HP03] and CDuce [BCF03], and the approaches that integrates native XML support in existing general programming languages (e.g. Java, C#), such as XJ [HRS$^+$05], XOBE [KL02], XTATIC [GP03], C*w* [BMS05] and XACT [KM06]. In general, XML transformations languages are usually used to transform an XML document into another, changing the vocabulary and reusing some content (i.e. data and XML fragments) from the original document. It is also possible to use for extending, filtering or modifying an XML document, maintaining the same vocabulary in the output. Moreover, they also come with support to statically type-checking. This versatility, however, comes at a cost: *(i)* it requires developers to be familiar with another language; *(ii)* they offer more programming options than an FOP programmer needs; and *(iii)* some languages define their own internal type system for XML data, which requires to define the mapping between XML-Schema and language types. However, the XAK approach facilities to the FOP domain and application engineers the definition of deltas and the validation of them by annotating the XML elements that can be modules at instance- and schema-level. The XAK composer is based on existing XML parser and the XAK validator gives precise localization of error messages, based on the XDuce type system and subtyping relationship.

XML Differences. XML deltas are closely related to the idea of XML differences. The main difference stems from their purpose, the former implement a feature in an SPL and are normally built separately while the latter are calculated as the difference between two XML documents. Among different approaches in the literature, *XML Patch* [Urp06], *XMLDiff [alp01, Cor02]* and *DeltaXML [LF01]* represents the differences by XML documents, whereas *X-Diff*, *diff-X*, represents the differences as an script that includes update operations.

Updates and Incremental XML Validation The composition of XAK and the validation of the XAK type-system are closely related to update operations and incremental validation of XML documents [KSR02, BHFA04, BML+04, PV03, SYK+10]. When a valid XML document will be updated in an XML DB, it has to be verified that the updated document still conforms to the imposed schema. To accept an update, the validity of the result is checked first (without any change on the original document). Validation tests are performed incrementally, i.e., only the validity of the part of the document directly affected by the update is checked. Changes to the original document are effectively performed only when the update is accepted.

The main difference of XAK with all of them is that 1) not all the element or attribute nodes can be updated. We select in two-levels: first, we define which type of elements can be updated in the schema and second, elements are identified by an ID instead of selecting by an XPath; 2) we derive an XML-Schema instead of creating a specific tool (e.g. an automata) for delta validation; and 3) we derive a set of constraints for each delta document, which requires them to the input document. All the above approaches embed these constraints in the derived algorithm or tool (e.g. an automata). For example, the SAXE system[KSR02] rewrite an Update-XQuery statement into a safe Update-XQuery statement by embedding constraint check sub queries into the update statement. However, the XAK type system define them separately from the tool in order to reuse in different contexts: *(i)* validating a single input document before composing or *(ii)* validating an entire SPL.

Object-oriented modularisation in HTML documents The need for systematic development techniques that scale with the increasing complexity of Web applications bring object-orientation, prototype-based, and componentware to the Web realm focusing on reuse in HTML document generation. Examples include *WebComposition Component Model* (WCCM)[GWG97], *Web Object Composition Model* (WOCM)[KNC01] and JESSICA project approaches[SWGZ00]. For example, JESSICA provides Object-oriented concepts such as abstraction, encapsulation, aggregation and inheritance through templates and page (i.e. object) creation by template instantiation in a Web setting. However, OO does not provide the right level of granularity for FOP requirements. Another drawback is that they provide a weak type system for content validation, which does not fully guarantee that the generated HTML document is a valid instance document, hence, errors are detected at execution time or ignore by the browser parser.

By contrast, this work strives to highlight the main differences between composition and traditional inheritance as a reuse technique, and addresses the peculiarities posed by markup languages as opposed to object-oriented ones, such as the importance of being a valid document. The rest of the paragraphs outlines these three approaches.templates can be derived by inheriting from other templates. Those templates are represented using an XML vocabulary.

Safe Composition in SPL Our work on type checking feature-oriented product lines was motivated by the work of Thaker et al. [TBKC07]. They suggested the development of a type system for feature oriented product lines that does not check all individual programs but the individual feature implementations. Influenced by this work, Delaware et al. [DCB09] and Apel et al. [AKGL10] define a constraint-based type system for Java-like feature-oriented languages: *Lightweight Feature Java* (LFJ) and *Feature Featherweight Java (FFJ)*, respectively. Both type systems check whether a given product specification falls into the subset of type-safe specifications described by the feature model. In other words, checking safe composition of a product line amounts to showing that the programs allowed by the feature model are contained within the set of type-safe products. Therefore, the goal of our tool is to complement the LFJ and FFJ type systems in order to type-checking Software Product Lines that contains XML documents. The XAK type system is also a constraint-based type systems in order to check whether the composed XML document will be contained within the set of valid documents with respect to the given XML-Schema.

3.9 Conclusions

This chapter addresses how feature-oriented programming can be extended to XML artefacts. The insights are realized through XAK, a language for XML delta composition. XAK offers an alternative way to modularise and refine code for languages where no other modularisation technique is available. Moreover, considerable effort has been devoted to obtain delta grammars so that deltas can be check out at the time they are defined. The importance of these insights rest on the importance of markup languages in the world of Web application development [RJ05].

Expressiveness and suitability have been the guiding principles throughout. Expressiveness is checked out through six case studies. Suitability is faced by delving into the complexities of delta validation and delta composition so that deltas are made *suitable* to the XML way of programming. However, other issues such as low coupling (i.e. deltas are not overly related with each other) or high cohesion (i.e. deltas are highly related internally) has not been addressed. Deltas are not totally independent entities. Dependencies might exist. These dependencies might force deltas to be composed in a certain order. Even more, delta behaviour might be affected by which other deltas they are composed with. This is an intriguing but very real problem. Assuming that applications can be gradually and monotonically created, starting with a prototype that is incrementally enriched with new features is not always possible. Even proponents of the Agile development recognize that enriching a prototype with a new functionality might lead to re-code functionality previously incorporated in the prototype in some ulterior development iterations (i.e. "scrums") [Mey14]. This important issue is left for future work.

Parts of the work described in this chapter have been previously presented:

- O. Díaz, F. I. Anfurrutia, and J. Kortabitarte. Using DITA for documenting Software Product Lines. In *Proceedings of the 9th ACM symposium on Document engineering (DocEng'09)*, 231-240, 2009.

- F. I. Anfurrutia, O. Díaz, and S. Trujillo. On refining XML artifacts. In *Proceedings of the 7th International Conference on Web Engineering (ICWE'07)*, volume 4607 of LNCS, pages 473–478. Springer, 2007.

- O. Díaz, S. Trujillo, and F. I. Anfurrutia. Supporting production strategies as refinements of the production process. In *Proceedings of the 9th International Software Product Line Conference (SPLC'05)*, volume 3714 of Lecture Notes in Computer Science, pages 210–221, Rennes, France, September 2005. Springer-Verlag.

Chapter 4

XML in Web development

4.1 Overview

The increasing growth in size and complexity of Web sites calls for a systematic way to web sites development that allows to face the stringent demands imposed on both the development and maintenance of these systems. Model-based approaches have been proposed to mitigate this situation. These approaches aim to find models, preferably orthogonal, that allow designers to declaratively specify a specific concern of the application without being immediately immersed in details of implementations. This chapter presents *XLeaflet*, a domain-specific language for web sites development. Each concern is separately described in an XML document. Moreover, a distinctive feature of *XLeaflet* is its architecture: thick-browser. This can account for an important reduction in the network traffic.

The chapter is organized along common Design Science activities: explicate problem, define requirements, artefact design, artefact implementation and artefact demonstration. Related work and conclusions end the chapter.

4.2 Problem Explanation

In which practice does the problem appear? There is tremendous pressure on Web developers to "code-and-publish". And the background of developers can be quite diverse with typically no experience in Software Engineering. They are guided by the features in the tools and language constructs. This free-form style of development can lead "to use ad-hoc, hacker-type approaches, which lack rigour, systematic techniques, sound methodologies, and quality assurance" [GM01]. And these practices can be disastrous as Web masters have to face maintenance, being a frustrating experience to see how often the website bottleneck slows and restricts the evolution of the organization the website is supposedly serving. Summing it up, website development is complex.

In Software Engineering, two principles to fight back complexity are abstraction and separation of concerns. Website development is not foreign to these practices. Indeed, traditionally HTML artefacts tend to mix together both content, rendering and layout within a single artefact: the HTML page. Now, good practices advice to decouple content (e.g. in a separated XML data document), and use transformation languages (e.g. XSLT) to indicate how this content is to be presented along HTML. In this setting, HTML does not hold content but its role is limited to dictate rendering directives. From this viewpoint, HTML is a Domain Specific Language (DSL) for content rendering. In this way, content is separated from rendering directives which are described through an HTML expression.

So far separation of concerns is limited to content and presentation. Other concerns of website construction are scattered around a diverse range of technologies and artefacts. This hinders maintainability and development since these concerns are expressed in terms of general programming language (e.g. JavaScript) difficult to debug and maintain.

This work pushes the "XSLT approach" to analyze the extent to which a *whole* website can be obtained. The vision is a **Domain Specific Language** (DSL) that declaratively permits no techies to create a website. DSLs are usually geared towards a specific domain or application, offering only a restricted suite of notations and abstractions. Hence, this DSL vision will not be possible in a general basis but it could happen for specific domains. Specifically, we look at static, content-oriented websites, hereafter referred to as ***"leaflet websites"***. Conference websites, product catalogues or websites with content about a teaching course are the kind of websites we are tackling. Here, the challenge rests more on rendering and navigation rather than supporting transactional-like functionality (e.g. purchases, enrolments and the like).

What is the problem and the negative consequences of not addressing it? The research question is

> *could "leaflet websites" be developed with the only help of a DSL?*

Conceiving website development as a DSL expression construction brings several benefits that could not be obtained otherwise:

- DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs,
- DSL programs are concise, self-documenting to a large extent, and can be reused for different purposes,
- DSLs enhance productivity, reliability, maintainability, and portability. Additionally, confidence in the correctness of the specifications is greatly improved when all the extra syntactic noise is eliminated from the specifications,
- DSLs embody domain knowledge, and thus enable the conservation and reuse of this knowledge.

Next section outlines a solution to the explicated problem in the form of an artefact and elicits requirements, which can be seen as a mutation of the problem into demands on the proposed artefact.

4.3 Requirement Definition

Our research question (i.e. *could "leaflet websites" be developed with the only help of a DSL?*) already conveys the kind of artefact to be developed: a DSL.

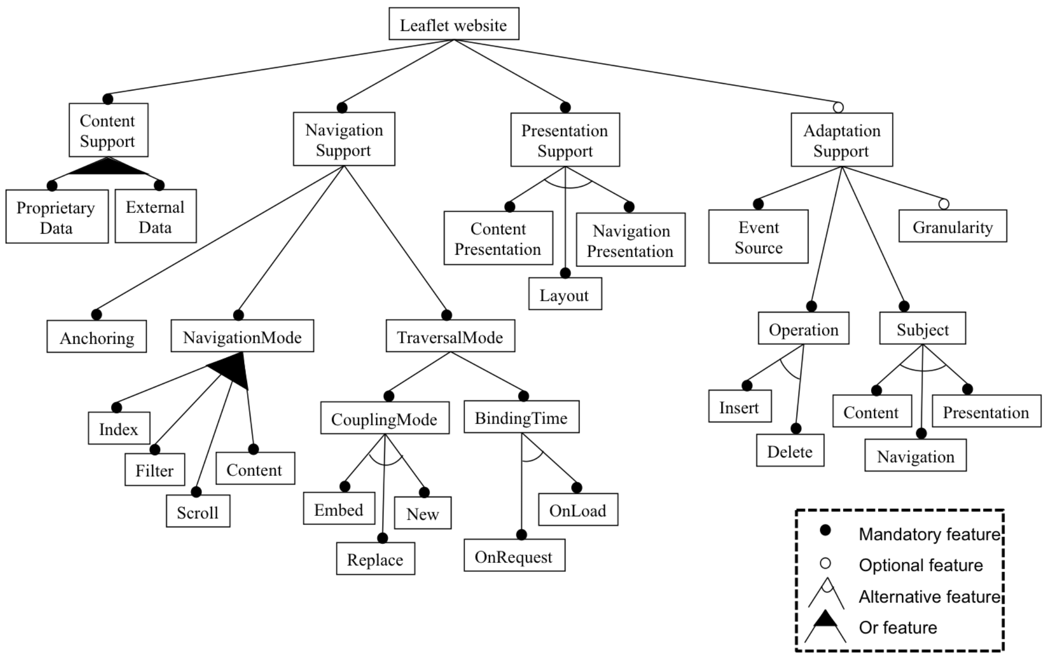


Figure 4.1: Feature-diagram. Different concerns are identified as impacting "leaflet website" development

For the purpose of this work, we focus on two requirements: expressiveness and performance.

4.3.1 Expressiveness

Expressiveness refers to the degree to which a set of constructs is capable of representing the entities of interest in a domain. Hence, the very first task is to characterize the target domain, i.e. *leaflet websites*. We characterize ***leaflet websites*** in terms of features. Figure 4.1 depicts the feature-diagram for *leaflet websites*. This diagram states that a "leaflet website" is developed by different concerns, such as content, navigation, presentation and adaptation. Next paragraphs introduce each feature:

- *Content Support*. Usually, a website unifies and presents content from heterogeneous data sources. Some data belongs to the website itself (i.e. *proprietary data*), whereas some other data is shared with other websites or applications (i.e. *external data*). Normally, the latter is held in a database, whereas the former is embedded in a *(leaflet) document*. This implies that a document (in the sense of an XML document) becomes the unit of delivery. The HTML page keeps on being the unit of presentation but the *document* becomes the unit of delivery. Instead of looking at a website as a set of pages, this view conceives a website as an inter-related set of documents. A document is brought to the browser, and once there, HTML pages are *dynamically* generated. A document is a self-contained chunk of information that is accessed/browsed as a unit,

- *Navigation Support*. This strives the fact that the average user session involves intensive browsing on a *(leaflet)* document. Browsing content of documents needs mechanisms for *anchoring* content in order to identify the origin and destination of a *link* navigation. Moreover, in case that the destination content is a large set of data distinct *navigation mode*s and *traversal mode*s can be needed. For *navigation mode*s, mechanisms such as *index* or *filter* are considered in order to reduce the set of data. A *scroll* have also been identified as effective mechanism in order to guide the browsing of the content. Finally, the *content* provides the set as is.
  In relation to a link *traversal mode*, it poses two questions: How are coupling the origin and target content of a link? When is traversed the link to show the target? At this respect, figure 4.1 depicts two features: *couplingMode* and *bindingTime*, respectively. Website wise, the target content can be *coupling* (showed) in the same page where the origin content resided (i.e. *embed*), *replace* the origin content or attached in a new window (i.e. *new*). The *bindingTime* can be executed automatically (i.e. *onLoad*), that is, the target is shown at the same moment that the origin, or it is postponed until a user event (i.e. *OnRequest*),

- *Presentation Support*. This describes the look and feel of both the content and the navigation elements, as well as the layout concerns,

- *Adaptation Support*. As web sites grow in size and complexity, mechanisms to both lightening cognitive overload and providing user guidance are in great demand. Both mechanisms can be realized by making websites adaptable. Adapting a web application requires to take decisions of *i) when* should be realized (i.e. *Event Source* and *Granularity*); *ii) what* operations are allowed (i.e. *Operation*); and *iii) which* concerns are changeable (i.e. *Subject*).

Therefore, constructs are needed to describe content, navigation, presentation and adaptation needs of the target application.

4.3.2 Performance

DSLs rise abstraction but might risk performance. That is, there exists a potential loss of efficiency when compared with hand-coded software. Speeding up development and maintenance (envisaged benefits of using a DSL) might not be at the cost of users suffering latency gaps. Hence, the artefact implementation should be performed. To this end, we impose the artefact architecture to be thick browser. Next we explain the rationales.

Most web applications commonly follow a thin-browser approach. Here, all of the application control resides on the server, and the browser is just used for rendering purposes. There is little control of the client's configuration. Most e-commerce applications use this architecture as it does not make good business sense to eliminate any sector of customers just because they do not have sufficient client capabilities [Con00]. Data-intensive applications are also good candidates for exhibiting this architecture. In these applications, data is not brought to the client but processed at the server, near where the data lies, and just the final outcome, the HTML page, is sent to the browser. On the other hand, a browser request will be required every time a new page is displayed. This could lead to an increase in network traffic, low site promptness and site bottlenecks.

By contrast, a thick-browser architecture attempts to mitigate this problem at the price of increasing the demands on the browser configuration. This feature allows a whole bulk of pages to be generated from one document with a single connection to the server. This can account for a sensible reduction on the network traffic as connections to the server are limited to requests for another document, invoking services or connecting to DBMS for extracting data. Navigation-intensive applications could be better served by this approach where the same content can be displayed several time without incurring in any communication cost.

The main concern of this work is to gain some insight into a thick-browser architecture for Web applications. Specifically, in this approach the additional processes performed by the browser are page generation, navigation and adaptation logic. The potential advantages are threefold: *i)* reduction in the network traffic, *ii)* enhancement of the site promptness, and *iii)* reduction in the server load. At the same time, the zero-deployment advantage is retained, i.e. code is centralized at the served and instantaneously deployed on the browser.

Next section addresses the design of *XLeaflet*, a DSL that addresses the explicated problem and strives to fulfil the previously defined requirements.

4.4 Artefact Design: XLeaflet

This section introduces *XLeaflet*, a DSL for leaftlet website development. This section tackles the expressiveness requirements set in subsection 4.3.1, i.e. means to describe the content, navigation, presentation and adaptation needs of the domain at hand. Next subsections address each of these concerns through four models: the content model, the navigation model, the presentation model and the adaptation model. A sublanguage is defined for each model. The DSL is illustrated with the help of a running example.

4.4.1 The content model

Concerning content and its structure, three issues are addressed: its nature, its source and its obsolescence. **Content nature** refers to whether you are handling data or documents [Bou03]. Data structure is characterized by being regular, fine-grained and where the order is often not significant. In a conference web-site, attendee data follows this pattern: the same data must be collected for each attendee, the granularity of this data is often atomic, and the order in which this data is arranged does not convey any meaning. By contrast, document structure commonly follows a irregular structure, larger-grained data and the order is significant. As an example, the introductory information about a conference, i.e. outline, topics of interest, distinct deadlines to authors or instructions to authors,

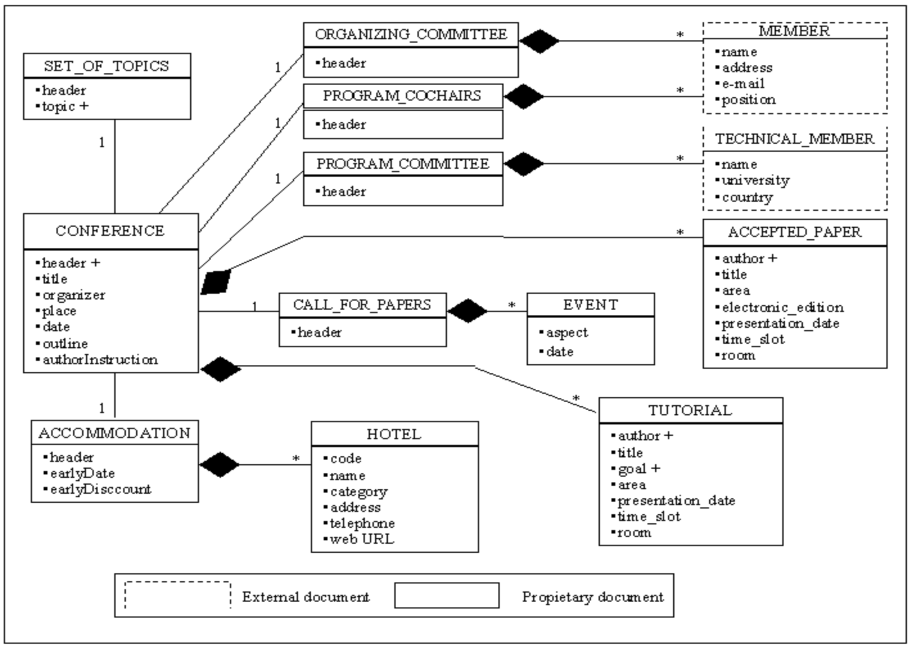


Figure 4.2: The content diagram for the conference example.

have an irregular structure. In order to accommodate the idiosyncrasies of each conference, these aspects do not follow a common, regular structure. For instance, a specific conference can have a boat trip or provide strict guidelines for author's manuscripts, whereas other conferences do not include any of these aspects. Even for the very same notion (e.g. the banquet) different conferences impose distinct structures (is there a reception previous to the banquet? should a map be included? is dress-up required?).

The XML specification holds the potential in devising effective integration of both data and document-centric sources and is being widely adopted as a standard for information representation and exchange. Hence, XLeaflet uses an XML document to support the content of a *web site*.

A UML class-diagram for the conference example is shown in Figure 4.2. A conference content includes data about the aims of the conference, the organizing and program committee, the call-for-papers, the list of accepted papers and so on.[1]

[1]Although the UML notation is used, it is worth noticing that the rectangles do not stand for classes nor entities. They refer to sub-documents, better said, XML elements (i.e.: tags) of the content document. How these elements have been ascertained from use-cases or why these

```
<?xml version="1.0"?>
<?xleaflet:navigation_document href="ifipNavigation.xml" type="text/xml"?>
<?xleaflet:adaptation_document href="ifipAdaptation.xml" type="text/xml"?>
<?xleaflet:presentation_document href="ifipPresentation.xml" type="text/xml"?>
<?xleaflet:connection_document href="ifipDBConnection.xml" type="text/xml"?>
<CONFERENCE xmlns:xleaflet="x-schema:../schemas/xleafletSchema.xml">
  <HEADER>International Federation for Information Processing </HEADER>
  <HEADER>IFIP 2.6 WORKING CONFERENCE ON DATABASE SEMANTICS (DS-9)</HEADER>
  <TITLE>SEMANTIC ISSUES IN e-COMMERCE SYSTEMS</TITLE>
  <ORGANIZER>by the IFIP Working Group 2.6 (Database)</ORGANIZER>
  ...
  <OUTLINE>E-commerce systems have been capturing the popular imagination ... </OUTLINE>
  <SET_OF_TOPICS>
    <HEADER>TOPICS OF INTEREST</HEADER>
    <TOPIC>Data modeling and query languages for databases for e-commerce</TOPIC>
     ...
   </SET_OF_TOPICS>
   <CALL_FOR_PAPERS>
    <HEADER>Call For Papers</HEADER>
    <EVENT>
      <NAME>Deadline for submissions of abstract</NAME>
      <DATE>November 30, 2000</DATE>
      <NEW image="hongkong-imagenes/newinfo.gif"/>
    </EVENT>
    ...
  </CALL_FOR_PAPERS>
  <AUTHOR_INSTRUCTIONS>Authors are invited to submit contrib...</AUTHOR_INSTRUCTIONS>
  <ORGANIZING_COMMITTEE>

    <HEADER>CONFERENCE ORGANIZATION</HEADER>
(a) | <xleaflet:QUERY rowName="MEMBER" connection="xmldemo" obsolescence="0">
    |   <xleaflet:SELECT> NAME, ADDRESS, EMAIL, POSITION</xleaflet:SELECT>
    |   <xleaflet:FROM>ORGANIZATION_MEMBERS</xleaflet:FROM>
    |   <xleaflet:WHERE>POSITION='Organization committee'</xleaflet:WHERE>
    |
    | </xleaflet:QUERY>
    <MEMBER>
      <NAME/>
      <ADDRESS/>
      <EMAIL/>
      <POSITION/>
    </MEMBER>
  </ORGANIZING_COMMITTEE>
  <PROGRAM_CO_CHAIR>
    <HEADER>PROGRAM CO-CHAIR</HEADER>
    <xleaflet:QUERY ... </xleaflet:QUERY>
    <MEMBER>...</MEMBER>
  </PROGRAM_CO_CHAIR>
  <PROGRAM_COMMITTEE>
    <HEADER>PROGRAM-COMMITTEE</HEADER>
    <xleaflet:QUERY ... </xleaflet:QUERY>
    <TECHNICAL_MEMBER>...</TECHNICAL_MEMBER>
  </PROGRAM_COMMITTEE>
  <ACCEPTANCE_RATE>80%</ACCEPTANCE_RATE>
  <ACCEPTED_PAPER>
    <AUTHOR>Kenneth R. Jacobs</AUTHOR>
    <TITLE> Innovation in Database Management: Computer Science vs Engineering.</TITLE>
    <AREA>Data Management</AREA>
    <ELECTRONIC_EDITION>innovation.pdf</ELECTRONIC_EDITION>
    <DATE>April 25, 2000</DATE>
    <ROOM>1</ROOM>
    <TIME_SLOT> 9:00 - 10:00 </TIME_SLOT>
  </ACCEPTED_PAPER>
  ...
</CONFERENCE>
```

Figure 4.3: The content document for the conference example.

The resulting content document is shown in Figure 4.3.

Content source. Web-site development strives to unify distinct and potentially heterogeneous data sources. According to this aspect, content can be characterized as proprietary or external. Proprietary content belongs to the web-site and it is provided within the content document itself. For instance, content about the aims of the conference, author instructions and the like are examples of proprietary content; it is not share with other sites. By contrast, external content is normally held in a database. A dotted line is used in Figure 4.2 to indicate this situation. As an example, ORGANIZING_COMMITTE members data is retrieved from a database. As shown in Figure 4.3a, an *XLeaflet* element, tagged `<xleaflet:QUERY>` is used for this purpose where the *connection* attribute holds a configuration XML file for connection purposes. At run-time, the processing of the `<xleaflet:QUERY>` element causes the attached query to be executed. This query returns an XML document having *MEMBER* as its root element (the *rowName* attribute) , and *NAME, ADDRESS* and *EMAIL* as its sub-elements. It is worth noticing that this element is interpreted only when the user navigates to the *MEMBER* element.

Content obsolescence. External data is brought to the browser. The *"obsolescence"* attribute of the `<xleaflet:QUERY>` element indicates the elapsed time that makes this content obsoleted. The values *"inf"* and *"0"* indicate whether refreshment should never occur or should take place every time this content is visited, respectively. Otherwise, this attribute holds the elapsed time that make this external data outdated. For our example, obsolescence is set to "0", indicating that data about the program committee should be loaded just once regardless on the times this data is visited. Recency requirements as well as the impact of retrieving outdated content should be considered to set the value of this attribute.

4.4.2 The navigation model

As shown in the previous section, *XLeaflet* supports "the content space" by means of an XML document. The navigation model addresses how this document can

elements have been chosen to be explicitly represented in the UML diagram among the potentially large set of elements of the content document is out of the scope of this work.

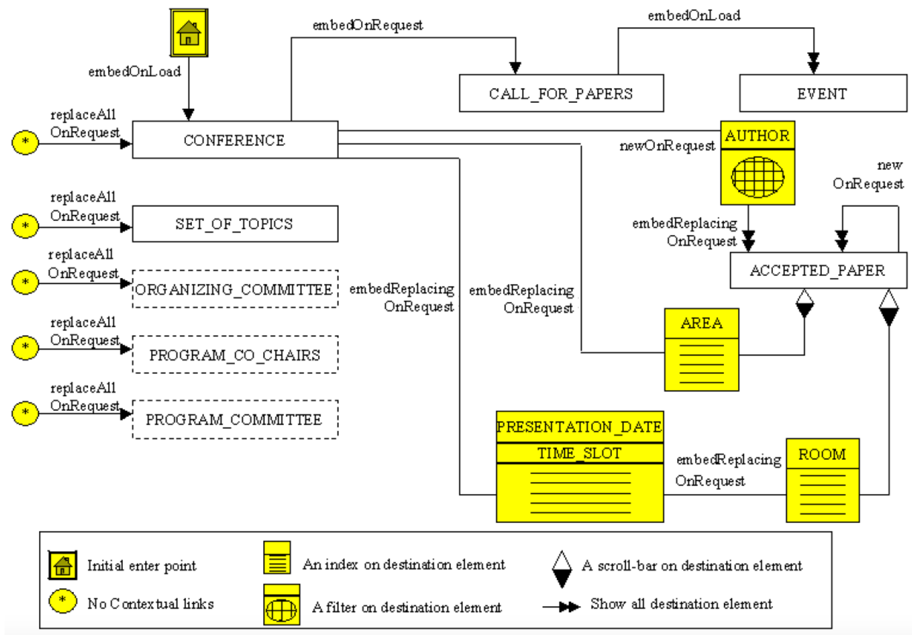


Figure 4.4: The navigation diagram for the conference example.

be traversed through links. Figure 4.4 shows part of the navigation diagram for the conference example where the different arrows represent the possible links along "the content space". A link has an origin and a destination anchor together with a navigation and a coupling mode. This diagram is realized in the navigation document (see Figure 4.5). Each arrow of the navigation diagram maps to a *<xlefatlet:LINK>* element in the navigation document.

In order to associate a navigation document with its content document, we follow a similar approach as the one used in XML. The processing instruction element "*<?xml:stylesheet ...?>*" attached to an XML document, indicates the XSLT document to be used for rendering the XML document. Likewise, *XLeaflet* provides three processing instruction elements, *<?xleaflet:navigation_document ...?>*, *<?xleaflet:presentation_document ...?>* and *<?xleaflet:adaptation_document ...?>*, to indicate the navigation, presentation and adaptation document associated with the content document, respectively (see the top of Figure 4.3).

In *XLeaflet*, link specification has to do with three issues: the anchor identification, the navigation mode and the coupling mode.

```
<?xleaflet:adaptation_document href="adaptConferencia.xml" type="text/xml"?>
<xleaflet:LINKS xmlns="x-schema:../schemas/xleafletSchema.xml">
  <!—home link–>
  <xleaflet:LINK title="Conference" from="home" to="/CONFERENCE">
    <xleaflet:CONTENT order="1" couplingMode="embedOnLoad"/>
  </xleaflet:LINK>
  <!—contextual links–>
  <xleaflet:LINK title="Call For Papers" from="/CONFERENCE" to="CALL_FOR_PAPERS">
    <xleaflet:CONTENT order="1" couplingMode="embedOnRequest"/>
  </xleaflet:LINK>
  <xleaflet:LINK title="Event" from="/CONFERENCE/CALL_FOR_PAPERS" to="EVENT">
    <xleaflet:CONTENT order="1" couplingMode="embedOnLoad"/>
  </xleaflet:LINK>
  <xleaflet:LINK title="Accepted papers" from="/CONFERENCE" to="/CONFERENCE">
    <xleaflet:CONTENT order="1" couplingMode="replaceAllOnRequest"/>
  </xleaflet:LINK>
(a) | <xleaflet:LINK title="Paper by Area" from="/CONFERENCE" to="ACCEPTED_PAPER">
    |   <xleaflet:INDEX order="1" title="Index of papers by area"
    |                              couplingMode="embedReplacingOnRequest">
    |     <xleaflet:INDEX_PROPERTY propertyName="AREA"/>
    |   </xleaflet:INDEX>
    |   <xleaflet:CONTENT order="2" couplingMode="embedReplacingOnRequest">
    |     <xleaflet:SCROLL first="yes" previous="yes" next="yes" last="yes"/>
    |   </xleaflet:CONTENT>
    |
    | </xleaflet:LINK>
 <xleaflet:LINK title="Paper by date, time-slot and room" from="/CONFERENCE"
                                                     to="ACCEPTED_PAPER">
    <xleaflet:INDEX order="1" title="Index by date and time slot:"
                              couplingMode="embedReplacingOnRequest">
      <xleaflet:INDEX_PROPERTY propertyName="DATE"/>
      <xleaflet:INDEX_PROPERTY propertyName="TIME_SLOT"/>
    </xleaflet:INDEX>
    <xleaflet:INDEX order="2" title="Index by room" couplingMode=" embedReplacingOnRequest ">
      <xleaflet:INDEX_PROPERTY propertyName="ROOM"/>
    </xleaflet:INDEX>
    <xleaflet:CONTENT order="3" couplingMode="embedReplacingOnRequest"/>
  </xleaflet:LINK>
(b) | <xleaflet:LINK title="Other papers of these authors" from="/CONFERENCE/ACCEPTED_PAPER"
    |            to="/CONFERENCE/ACCEPTED_PAPER[AUTHOR=$FROM/AUTHOR]
    |                                            [TITLE !=$FROM/TITLE]">
    |   <xleaflet:CONTENT order="1" couplingMode="newOnRequest"/>
    |
    | </xleaflet:LINK>
  <!– No contextual links –>
  <xleaflet:LINK title="Conference" from="*" to="/CONFERENCE">
    <xleaflet:CONTENT order="1" couplingMode="replaceAllOnRequest"/>
  </xleaflet:LINK>
  <xleaflet:LINK title="Topics of interest" from="*" to="/CONFERENCE/SET_OF_TOPICS">
    <xleaflet:CONTENT order="1" couplingMode="replaceAllOnRequest"/>
  </xleaflet:LINK>
  <xleaflet:LINK title="Conference Organization" from="*"
                                   to="/CONFERENCE/ORGANIZING_COMMITTEE">
    <xleaflet:CONTENT order="1" couplingMode="replaceAllOnRequest"/>
  </xleaflet:LINK>
  <!–other links, similar to previous are omitted for spacing –>
(c) | <xleaflet:LINK title="Paper by Author" from="/CONFERENCE" to="ACCEPTED_PAPER">
    |   <xleaflet:FILTER order="1" title="Search papers by author" couplingMode="newOnRequest">
    |     <xleaflet:SEARCH_PROPERTY title="Author" propertyName="AUTHOR" predicate="belongsTo"/>
    |   </xleaflet:FILTER>
    |   <xleaflet:CONTENT order="2" couplingMode="embedReplacingOnRequest"/>
    |
    | </xleaflet:LINK>
</xleaflet:LINKS>
```

Figure 4.5: The navigation document for the conference example.

Anchor identification. The distinct element tags of the XML content document are conceived as potential anchors from which to define origins and destinations. Elements within the XML content document are addressed using the W3C standard XPath notation[CD99]. XPath conceives an XML document as a tree of nodes, and follows a notation similar to the UNIX directory paths to address each node within the document (a *location path* using XML parlance). We have extended this notation in distinct ways to accommodate some navigation requirements. We regard four kinds of location paths: relative location paths, absolute location paths, the non-contextual location path, and the 'home' location path. The former two are found in XPath and the main difference stems from what is the origin of the path: the *context node* (i.e. the current node being visited) in the case of relative location paths, or the root of the document (denoted by "/") for absolute location paths. This notation allows to locate the descendants of a given node (either the current node or the root node), and in so doing, supports forward navigation. However, backward navigation is also useful, i.e. the location of the predecessors of the context node. We have then introduced the ".." syntax. As an example, consider we are currently processing an ORGANIZING_COMMITTE node. If from this node we want to reach a sibling node(e.g.: a PROGRAM_COMMITTEE node), the link destination path would be *"../PROGRAM_COMMITTEE"*. The non-contextual location path stands for any path, and supports non-contextual links, i.e. links that are accessible regardless of the current context node. This path is denoted by "*" and can appear only as value of a link origin. Finally, the home location path is used to denote the origin of those links which constitute the entry points of the navigation. This path is denote by *"home"*.

Figure 4.4 shows a possible navigation diagram for the conference example. As indicated by the *home* location path, the CONFERENCE outline is readily presented once connected to the site. Several no-contextual links are always available to display the ORGANIZING_COMMITTEE, the PROGRAM_COMMITTE or the SET_OF_TOPICS within the conference scope. By contrast, the CALL_FOR _PAPERS can only be accessed while being on the conference outline, which in turn leads the way to the EVENTs.

Each link in this diagram is mapped to a `<xleaflet:LINK>` element in the navigation document (see Figure 4.5). As an example, consider the link that goes from a *CONFERENCE* node to its related *CALL_FOR_PAPER* nodes. This is expressed in *XLeaflet* as follows:

```
<xleaflet:LINK title="Call for Papers"
               from="/CONFERENCE"
               to="CALL_FOR_PAPERS"> ...
</xleaflet:LINK>
```

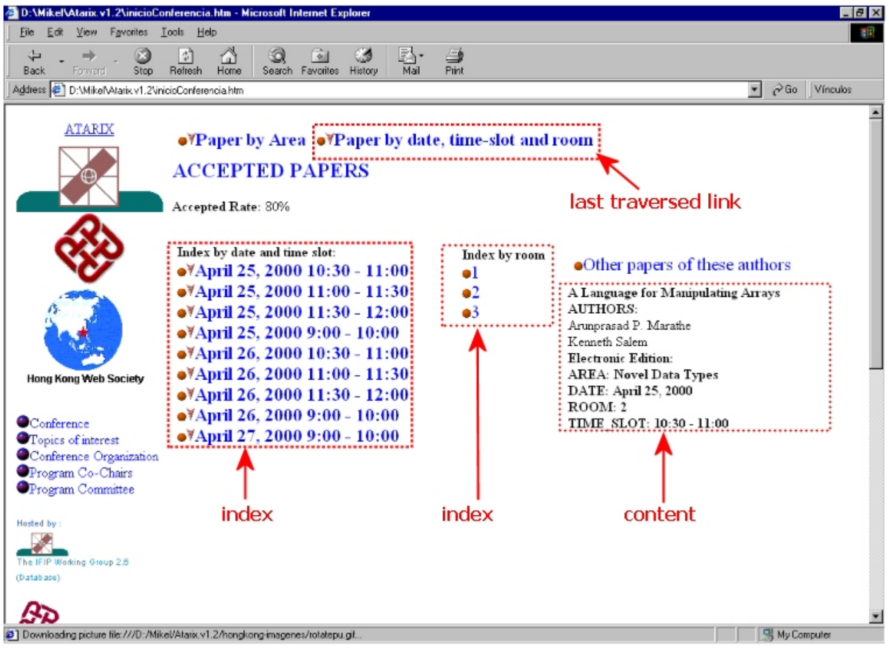
<xleaflet:LINK> is an XML element of the *XLeaflet* vocabulary. This element has a set of attributes which describe (1) the label of the link when rendered on the screen (the *title* attribute); the origin of the link (the *from* attribute) that indicates when the link is available (in this case the link is available when the *CONFERENCE* element is rendered); and the destination of the link (the *to* attribute) which states the element to be rendered when this path is followed. Notice that the *from* attribute holds an absolute path whereas the *to* attribute is a relative path, i.e. it refers to the accepted papers which hang on a concrete *CONFERENCE* node.

It is worth noticing how recursive links can be defined. For instance, the link that goes from an ACCEPTED_PAPER to other accepted papers by the same authors is specified as follows:

```
<xleaflet:LINK title="Other papers of these authors"
               from="/CONFERENCE/ACCEPTED_PAPER"
               to="/CONFERENCE/ACCEPTED_PAPER
                    [AUTHOR=$FROM/AUTHOR] [TITLE!=$FROM/TITLE]">...
</xleaflet:LINK>
```

Both, the *from* and *to* attributes hold an absolute path. However, the *to* path just addresses those *ACCEPTED_PAPER* elements which have the same *AUTHOR* as (and distinct *TITLE* from) the accepted paper being visited. At run time, the variable *$FROM* keeps the last *ACCEPTED_PAPER* node which is being visited.

Navigation mode. Navigation proceeds from a node of the XML document to the destination nodes of the chosen link. If a single destination node is available,



D:\Mikel\Atarix.v1.2\inicioConferencia.htm - Microsoft Internet Explorer

Address D:\Mikel\Atarix.v1.2\inicioConferencia.htm

ATARIX

●Paper by Area ●Paper by date, time-slot and room

ACCEPTED PAPERS

Accepted Rate: 80%

Hong Kong Web Society

Index by date and time slot:
- April 25, 2000 10:30 - 11:00
- April 25, 2000 11:00 - 11:30
- April 25, 2000 11:30 - 12:00
- April 25, 2000 9:00 - 10:00
- April 26, 2000 10:30 - 11:00
- April 26, 2000 11:00 - 11:30
- April 26, 2000 11:30 - 12:00
- April 26, 2000 9:00 - 10:00
- April 27, 2000 9:00 - 10:00

Index by room
- 1
- 2
- 3

●Other papers of these authors

A Language for Manipulating Arrays
AUTHORS:
Arunprasad P. Marathe
Kenneth Salem
Electronic Edition:
AREA: Novel Data Types
DATE: April 25, 2000
ROOM: 2
TIME_SLOT: 10:30 - 11:00

- Conference
- Topics of interest
- Conference Organization
- Program Co-Chairs
- Program Committee

Hosted by :
The IFIP Working Group 2.6
(Database)

Downloading picture file:///D:/Mikel/Atarix.v1.2/hongkong-imagenes/rotatepu.gif...

File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Go Vínculos
My Computer
last traversed link
index
index
content

Figure 4.6: Navigation from *Conference* to *acceptedPaper*: an index hierarchy along accepted papers.

navigation is straightforward. However, one-to-many navigation is found if an XML element encompasses a set of elements of the same type. The navigation mode indicates how to proceed in this case.

As an example, a CONFERENCE element can include a set of ACCEPTED_ PAPER sub-elements. When traversing a link from conference to its accepted papers, should all of then be processed at once? or is it preferably to browse then one by one? In this case, the designer is confronted with the decision of how the set of papers are traversed. This navigation mode is described by means of sub-elements of the `<xleaflet:LINK>` element.

Based on the WebML modeling language [CFB00], *XLeaflet* supports indexes, filters, and scrolls. A link can sequence some of these constructors to build up an aggregate navigation mode.

An index provides a shortcut to reach the desired destination nodes. This is supported through the `<xleaflet:INDEX>` element. For instance, to reach *ACCEPTED_PAPER* nodes from a *CONFERENCE* node, the next link is defined:

```
<xleaflet:LINK title="Paper by date, time-slot and room" from="/CONFERENCE"
                                        to="ACCEPTED_PAPER">
   <xleaflet:INDEX order="1" title="Index by date and time slot"
                     couplingMode="embedReplacingOnRequest">
      <xleaflet:INDEX_PROPERTY propertyName="DATE"/>
      <xleaflet:INDEX_PROPERTY propertyName="TIME_SLOT"/>
   </xleaflet:INDEX>
   <xleaflet:INDEX order="2" title="Index by room"
                    couplingMode="embedReplacingOnRequest">
      <xleaflet:INDEX_PROPERTY propertyName="ROOM"/>
   </xleaflet:INDEX>
   <xleaflet:CONTENT order="3" couplingMode="embedReplacingOnRequest"/>
</xleaflet:LINK>
```

The *<xleaflet:INDEX_PROPERTY>* sub-element indicates which property of the destination element (i.e. *ACCEPTED_PAPER*) serves as an index. By property is meant either an attribute or a sub-element. In the previous example, the system offers an index by the aggregation of *DATE* and *TIME_SLOT* (both sub-elements of *ACCEPTED_PAPER*) on its way to *ACCEPTED_PAPERs.* If the set of paper within a *date+timeSlot* is still large, you can index the resulting subsets by another concept. For instance, the previous example offers an index hierarchy where the second index arranges papers by *"presentationRoom"*. A possible layout of the previous navigation is shown in Figure 4.6.

If the set of destination nodes is large, a filter could be more suitable. This construct provides a query capability to restrict the set of destination nodes: edit fields are provided for inputting values used for searching along the set of destination nodes. The *<xleaflet:SEARCH_PROPERTY>* sub-element indicates a field for inputting a value. This sub-element has three attributes which describes: (1) the label of the field when rendered on the screen (the *title* attribute); the property of the destination element to search for (the *propertyName* attribute); and the predicate to be satisfied by the destination nodes in order to be rendered (the *predicate* attribute). *Predicate* can hold three values: *"belongsTo",* in which case the introduced value must be one of the values of the *propertyName* element; *"between"* in which case the user provides the two ends of an interval within which the value of searched property must be; and *"like"* in which case the user provides a pattern (i.e. a regular expression) to compare with the value of the searched property. An example for this navigation mode is given in Figure 4.5c).

If the set of destination nodes is low, the designer can choose to traverse the whole set right away. The `<xleaflet:CONTENT>` element indicates this alternative. Notice, that any traversal should end in a content mode since this is the only way to reach the nodes' content. An alternative is to show the destination nodes one by one rather than the whole set together. In this case, the `<xleaflet:CONTENT>` element has a `<xleaflet:SCROLL>` sub-element. An example is given in Figure 4.5a):

```
<xleaflet:LINK title="Paper by Area" from="/CONFERENCE"
                                    to="ACCEPTED_PAPER">
   <xleaflet:INDEX order="1" title="Index of papers by area"
                    couplingMode="embedReplacingOnRequest">
      <xleaflet:INDEX_PROPERTY propertyName="AREA"/>
   </xleaflet:INDEX>
   <xleaflet:CONTENT order="2" couplingMode="embedReplacingOnRequest">
      <xleaflet:SCROLL first="yes" previous="yes" next="yes" last="yes"/>
   </xleaflet:CONTENT>
</xleaflet:LINK>
```

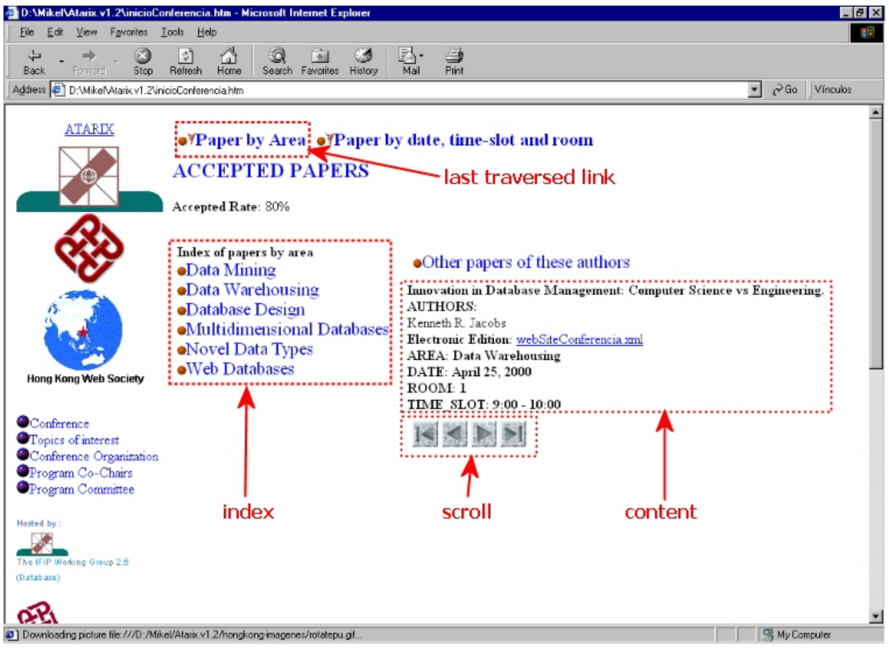
An index by *AREA* is first provided. Once an area is selected, a scroll allows to browse along the papers on this area. Figure 4.7 shows a possible rendering for this navigation mode.

Coupling mode. The coupling mode addresses how indexes, filters and scrolls should be arranged during link traversal. Should the destination node be displayed together with the origin node (*embed*), substitute the origin node (*replace*) or be displayed in a separate window (*new*)? Moreover, should this navigation occurs *on request* or take place automatically *on load*? These concerns appear also in the W3C XML Linking Language (XLink) proposal [DMOT01]. As these aspects are not completely orthogonal, we decide to mix both and provide the following coupling mode values: *newOnLoad, newOnRequest, embedOnLoad, embedOnRequest*[2], *embedReplacingOnRequest*[3], *embedReplacingOnLoad, replaceOneOnRequest* and *replaceAllOnRequest*[4]. If the navigation mode is aggregated, each nav-

[2]The *embed* refers to show the target node in the same page as the origin node

[3]The *embedReplacing* is similar to the one as before, but previous target nodes are replaced with the last one

[4]With the *replace* options, the designer can choose whether the substitution affects to all the


D:\Mikel\Atarix.v1.2\inicioConferencia.htm - Microsoft Internet Explorer

Address D:\Mikel\Atarix.v1.2\inicioConferencia.htm

ATARIX

Paper by Area Paper by date, time-slot and room

ACCEPTED PAPERS

Accepted Rate: 80%

Index of papers by area
- Data Mining
- Data Warehousing
- Database Design
- Multidimensional Databases
- Novel Data Types
- Web Databases

Other papers of these authors

Innovation in Database Management: Computer Science vs Engineering.
AUTHORS:
Kenneth R. Jacobs
Electronic Edition: webSiteConferencia.xml
AREA: Data Warehousing
DATE: April 25, 2000
ROOM: 1
TIME_SLOT: 9:00 - 10:00

Hong Kong Web Society

- Conference
- Topics of interest
- Conference Organization
- Program Co-Chairs
- Program Committee

Hosted by :
The IFIP Working Group 2.6
(Database)

Downloading picture file:///D:/Mikel/Atarix.v1.2/hongkong-imagenes/rotatepu.gif...

File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Go Vínculos
last traversed link
index
scroll
content
My Computer

Figure 4.7: An alternative way to support the navigation from *Conference* to *acceptedPaper*: an index on *area* and then, scroll along the papers of a selected area.

igation primitive specifies the coupling mode with respect to the previous stage. For instance, the screen dump shown in Figure 4.6 delivers the origin element (i.e. a *CONFERENCE* element) in the same page from both the indexes and the destination elements (i.e. an ACCEPTED_PAPER element). Another alternative would be to present both conference data and the indexes separately, in a different screen, from the selected ACCEPTED_PAPERs. The navigation mode is the same; the coupling modes are different. In this way, the designer is free to decide how tightly she wants to distribute the content.

4.4.3 The presentation model

Both content and link definitions have a presentation counterpart which addresses the look and feel of the final layout. This includes selecting the delivering mechanism (i.e fonts, background, icons and the like) for both content and links, as well as the distribution of these items along the presentation space (e.g. a page).

origin nodes (option *all*) or it is restricted to the last origin node being visited (option *one*).

```
<xleaflet:template matchType="LINK_TO_CONTENT">
   <IMG src="../links-imagenes/punto.gif" width="10" height="10"/>
   <xleaflet:NAVIGATOR>
      <A class="toContent">
         <xleaflet:value_of select="@title"/>
      </A>
   </xleaflet:NAVIGATOR>
</xleaflet:template>
<xleaflet:template matchType="LINK_TO_CONTENT"
                    matchInstance="/CONFERENCE/CALL_FOR_PAPERS">
   <xleaflet:NAVIGATOR>
      <INPUT type="button">
         <xsl:attribute name="value">
            <xleaflet:value_of select="@title"/>
         </xsl:attribute>
      </INPUT>
   </xleaflet:NAVIGATOR>
</xleaflet:template>
```

Figure 4.8: The default template for single links and how this template is overridden for links which stay in the context of *"Call for Papers"* content.

Delivering mechanism. One of the most often cited advantages of XML is the separation between content and presentation achievable through XSLT style sheets[Cla99]. An XSLT style sheet includes one or more *templates*, each of which contains the information for displaying a particular branch of the element hierarchy in the XML document. The `match` attribute identifies the specific branch by using an *XPath* expression. *XLeaflet* uses XSLT. However, the XSLT `match` attribute locates the specific branch based on the tags of the elements (i.e. the *XPath* expression) while *XLeaflet* also needs to locate elements by `type`. A `type` indicates the role played by this element during rendering, that is, whether the element supports content, single links, indexes, filters or scrolls. Hence, an `<xleaflet:template>` construct is introduced. *XLeaflet* templates follow the same pattern than XSLT templates, but now elements can be located by both its tag name as XSLT does, or its type.

The motivation for such distinction is that it allows the designer to specify a generic presentation for *XLeaflet* types (e.g. *link_to_content, index, scroll*) that can be later overridden for specific elements playing this roles. For instance, Figure 4.8 shows the default template for single links *(matchType="LINK_TO_*

```
<xleaflet:template matchType="PAGE_CONTAINER">
  <html>
     <head>
        <style type="text/css">
           .index { color: red; font-size:16pt; font-weight: bold; cursor: hand; }
           .home { color:white; font-size:16pt; text-decoration: none; }
           .contextual {color:#3399cc; font-size:16pt; text-decoration: none; }
           .noContextual { color:#3399cc; font-size:12pt; text-decoration: none; }
           A:visited { color:#CCCCCC; text-decoration:none; }
           A:hover { text-decoration:underline; }
           BODY{background:#ecfbff; ]
        </style>
     </head>
     <body>
        <table width="100%" height="100%">
           <tr height="100%">
              <td vAlign="top" width="170">
                 <center>
                    <a href="http://sipl68.si.ehu.es/xleaflet/">XLEAFLET</a>
                 <img height="87" src="hongkong-imagenes/logo.gif" width="170"/>
                    ...
                 </center>
                 <br/>
                   <xleaflet:apply_templates select="NO_CONTEXTUAL_
                                                    NAVIGATION_SET"/>
                 ...
              </td>
              <td width="4"/>
              <td valign="top" width="100%">
                <xleaflet:for_each select="INDEX | FILTER |
                                          CONTENT_CONTAINER">
                   <xleaflet:apply_templates select="."/>
                </xleaflet:for_each>
              </td>
           </tr>
        </table>
     </body>
  </html>
</xleaflet:template>
```

Figure 4.9: The general page of the conference site.

```
<xleaflet:template matchType="CONTENT" matchInstance="/CONFERENCE">
   <!–content omitted –>
   <h2 style="color:blue">Outline</h2>
   <xleaflet:value_of select="$matchInstance/OUTLINE"/>
   <h2 style="color:blue">Author instructions</h2>
   <xleaflet:value_of select="$matchInstance/AUTHOR_INSTRUCTIONS"/>
</xleaflet:template>
<xleaflet:template matchType="CONTENT" matchInstance="/CONFERENCE"
                    matchContext="xleaflet:LINK[@title='Accepted papers']">
   <h2 style="color:blue">ACCEPTED PAPERS</h2>
   <b>Accepted Rate: </b>
   <xleaflet:value_of select="$matchInstance/ACCEPTANCE_RATE"/>
</xleaflet:template>
```

Figure 4.10: The default template for the content type and how this template is overridden when a link is traversed.

CONTENT”). This rendering can be overridden for specific links at one’s discretion. Overridden takes places in two ways:

1. indicating the element’s relationship with an instance of the content document (the `matchInstance` attribute). For example, figure 4.8 illustrates how the single link presentation is overridden for those links ending at *“/CONFERENCE/CALL_FOR_PAPERS”*.

2. indicating the link which has brought about the navigation to this element (the `matchContext` attribute). For example, figure 4.10 shows how the conference presentation is overridden when the linked titled *“Accepted papers”* is traversed.

Distribution concerns. It refers to how items are arranged in a page. A stepwise process is used similar to JSP [Gea00]. We begin by designing a *general page layout* to be followed by any page of the site. This layout provides a first skeleton where some general canvas are drawn (e.g. the header, the body and the footer). The content of these canvas is decided separately in another style sheet which in turn, can left some “smaller” canvas for further refinement. Unlike the style sheets used for presentation purposes, these sheets are only concerned with distribution. Hence, they are referred to as style-free style sheets[VdV00]. The general page layout style sheet for the conference site is shown in Figure 4.9.

Besides this general layout, separate layouts are provided for each kind of item to be displayed, namely, content, single links, indexes, filters and scrolls. The user can override these default layouts and provide their own. Overriding is provided at the instance level, that is you can override how a given index or paper is distributed on the canvas, and left the rest of the instances follow the default layout.

4.4.4 The adaptation model

Previous models provide a "frozen" view of a site, that is the range of data, links and layouts that build up the site are fixed. However, this view could need to be adapted dynamically based on the current click-stream behaviour.

As web sites grow in size and complexity, mechanisms to both lightening cognitive overload and providing user guidance are in great demand. Both mechanisms can be realized through adaptation. The former by defining some precedence order among content/links so that some content/link can not be presented till some other content/link has already been visited. As an example, consider the conference site. It could be the case that accommodation information should be displayed if the conference outline has already been consulted. Otherwise, there is no point in cluttering the page with accommodation information till there is not signs the user is interested in attending it. Such interest can be measured by the type of content visited or the time spent on the site. On the other hand, user guidance could lead to adapt both the navigation space and the presentation based on previous interactions. For instance, consider that a large number of papers have been accepted. The list of papers is presented in a standard way. If the user shows interest in any of then (by clicking on it), the system can improve user guidance by changing the presentation of the related papers (i.e. those having similar key words) on the fly. A more flashy presentation will guide the user to those potentially interesting papers. This kind of mechanism offers potential for cross-selling where one purchase can cause either to highlight or to provide links to related products.

An adaptation model should indicate, first, criteria in which the adaptation is based and second, the subject of the adaptation, i.e. what can be adapted. In our model, the content, the navigation or the presentation documents are the subjects to be adapted based on the date and click behaviour obtained at run time.

Adaptation is defined in terms of the well-known event-condition-action rule paradigm. The rule is triggered by a given event; once triggered, the condition is considered, and if the condition is true then the action is executed. *Browser rule*s are described using an XML syntax, where the *<xleaflet:ON>*, *<xleaflet:IF>* and *<xleaflet:DO>* elements are introduced to describe the event, the condition and the action part of the rule, respectively.

An event occurrence is produced by the traversal of a link, either after or before the traversal takes place. The link is defined in terms of the link properties, using a XPath-like syntax. For instance, the event definition

```
<xleaflet:ON when="after"
             source="xleaflet:LINK[@to='/CONFERENCE/CALL_FOR_PAPERS']"/>
```

rises an event occurrence after traversing **any** link which ends at a *'CONFERENCE/CALL_FOR_PAPERS'* node. Another example would be:

```
<xleaflet:ON when="before" source="xleaflet:LINK[@from='home']"/>
```

rises an event occurrence before loading the very first page.

An event occurrence has three parameters: *$FROM,* which holds an XPath expression pointing to the origin node; *$TO,* which holds the arrival node; and *$DATE* from where the date can be retrieved. The *$TO* parameter is empty for *before* events.

Conditions evaluate predicates on event parameters or document content. The boolean expression can be cumbersome to specify as an XML syntax is used. Additional functionality can also be used by using *$FUNCTION:* key before referencing it. This functionality must be provided as a Javascript library and it will be invoked by the *XLeaflet* interpreter. As an example, the next condition:

```
<xleaflet:IF>
   <xleaflet:COMPARATION operator="smaller">
      <xleaflet:OPERAND>
       <xleaflet:CONSTANT type="int" value="$FUNCTION:dateDiference('$DATE',
          '$FUNCTION:getNodeValue(&quot;/CONFERENCE/CALL_FOR_PAPERS/
             EVENT[NAME=&apos;Deadline for submission of the paper&apos;]/
             DATE&quot;)')"/> a
      </xleaflet:OPERAND>
      <xleaflet:OPERAND>
         <xleaflet:CONSTANT type="int" value="0"/>
      </xleaflet:OPERAND>
   </xleaflet:COMPARATION>
</xleaflet:IF>
```

[a] &apos; and &quot; are reusable entities. They are used in XPath expressions when the ' or " characters causes a problem, respectively.

evaluates whether the deadline for paper submission is over by the time the event happens.

Finally, the action part indicates how the content, navigation or presentation documents are adapted. That is, the additions or removals of content, links or templates. As an example, the next action:

```
<xleaflet:DO>
   <xleaflet:INSERT_LINK parents="/xleaflet:LINKS">
      <xleaflet:LINK title='$FUNCTION:getNodeValue("$TO/TITLE")'
                     from="*" to="$TO">
         <xleaflet:CONTENT order="1" couplingMode="replaceAllOnRequest"/>
      </xleaflet:LINK>
   </xleaflet:INSERT_LINK>
</xleaflet:DO>
```

adds a link to the navigation space. The *<xleaflet:INSERT_LINK>* element introduces a new *<xleaflet:LINK>* element. It indicates the parent node from which this new link should hang through the "*parents*" attribute (e.g. *"/xleaflet:LINKS"*). Notice how the event parameter *$TO* is used to collect some information (from the last traversed *link*) for the construction of this new link.

Besides the `<xleaflet:ON>`, `<xleaflet:IF>` and `<xleaflet:DO>` elements, a rule has also two attributes: `granularity` and `enabled`. The former indicates whether a rule should be fired every time the associated event is risen (*"for_each_event"*) or just once during the current session (*"for_each_session"*). It is worth noticing that whereas database triggers' effects are persistent since they act on the database, the subject matter of browser rules is the session itself. Therefore, their effects vanish once the session is over. Hence this mechanism offers a session-based adaptability and it is complementary with server rules whose effects have a longer lifespan.

On the other hand, the `enabled` property indicates whether the rule is enabled or not. This attribute supports temporal deactivation of rules.

Next subsections present how browser rules can be used for both lightening cognitive overload and user guidance.

4.4.4.1 Lightening cognitive overload

Lightening cognitive overload can be achieved by *"tuning the navigation space"*. Although the available links are described during navigation modeling, this navigation space could need to be adapted at run-time. The designer should not be forced to define all links as readily available. This could lead to cluttered pages. Instead, the designer should distinguish between *immediate* links and *deferred* links. Both are defined in the very same way, but whereas the former are readily available in the navigation document, deferred links are subject to the happening of an event. In this way, the navigation space can be expanded (more links are added) or shrunk (some links are removed), and in so doing, the scope of the navigation is kept within cognitive thresholds.

As an example, consider the conference site. It could be the case that accommodation information should be displayed after the conference outline has already been consulted. Otherwise, there is no point in cluttering the page with accommodation information till there is not signs the user is interested in attending it. Such interest can be measured by the type of content visited or the time spent on the site. Let's consider that visiting the conference topics is regarded as a sign of interest in the conference. The rule in Figure 4.11 achieves this effect.

```
<xleaflet:RULE granularity="for_each_session" enabled="yes">
   <xleaflet:ON when="after"
               source="xleaflet:LINK[@to='/CONFERENCE/SET_OF_TOPICS']"/>
   <xleaflet:IF>
      <xleaflet:TRUE/>
   </xleaflet:IF>
   <xleaflet:DO>
      <xleaflet:INSERT_LINK parents="/xleaflet:LINKS">
         <xleaflet:LINK title="Accomodation"
                        from="*" to="/CONFERENCE/ACCOMMODATION">
            <xleaflet:CONTENT order="1" couplingMode="replaceAllOnRequest"/>
         </xleaflet:LINK>
      </xleaflet:INSERT_LINK>
   </xleaflet:DO>
</xleaflet:RULE>
```

Figure 4.11: Adaptation browser rule: inserting a link in the navigation document

The rule's event is associated to the traversal of **any** link leading to *"/CONFERENCE/SET_OF_TOPICS"*. If this event is risen, the rule's action extends the navigation space with a link to the conference's hotels which will be visible when the user returns to the home page. It is worth noticing that this rule should be fired just once, regardless of the times topics information is visited. Since, the rule's granularity is set to *'for_each_session'*.

4.4.4.2 Guiding the user

Guidance tactics include ascertaining potential useful shortcuts for the user or highlighting some content. As an example of providing useful shortcuts, consider that a non-contextual link is added on the fly for each paper that has been visited. The user can spend some time on locating a paper, browsing through distinct indexes. Once the paper is located, we would like to speed up future references through a non-contextual link that readily leads to the visited paper without cumbersome browsing. The rule in Figure 4.12 provides the required behaviour.

As an example of highlighting, consider that a large number of papers have been accepted. The list of papers is presented in a standard way. If the user shows interest in any of then (by clicking on it), the system can improve user guidance by changing the presentation of the related papers (i.e. those having the same

```
<xleaflet:RULE granularity="for_each_event" enabled="yes">
   <xleaflet:ON when="after"
            source="xleaflet:LINK[@to='/CONFERENCE/ACCEPTED_PAPER']"/>
   <xleaflet:IF>
      <xleaflet:TRUE/>
   </xleaflet:IF>
   <xleaflet:DO>
      <xleaflet:INSERT_LINK parents="/xleaflet:LINKS">
         <xleaflet:LINK title='$FUNCTION:getNodeValue("$TO/TITLE")'
                        from="*" to="$TO">
            <xleaflet:CONTENT order="1" couplingMode="replaceAllOnRequest"/>
         </xleaflet:LINK>
      </xleaflet:INSERT_LINK>
   </xleaflet:DO>
</xleaflet:RULE>
```

Figure 4.12: Adaptation browser rule: providing a shortcut link to accepted papers

key words) on the fly. A more flashy presentation will guide the user to those potentially interesting papers. The rule in Figure 4.13 achieves this effect.

4.5 Artefact implementation: supporting XLeaflet

XLeaflet is a DSL for *leaflet-website* development. Broadly, a *XLeaflet* expression stands for a whole website. So far, we have addressed the DSL syntax but not its operational semantics, i.e. how DSL constructs are going to be interpreted or compiled. This operational semantics conveys "the meaning" of the DSL constructs in terms of its effects. In addition, the implementation also embodies the strategies to address main non-functional requirements. Performance is a case in point.

The XLeaflet interpreter accounts for a thick-browser architecture to outperform traditional thin-browser solutions based on general programming languages. Here, all of the application control resides on the server, the browser is just used for rendering purposes, and a browser request is required every time a new page is displayed. This could lead to an increase in network traffic and low site promptness. By contrast, a thick-browser architecture moves to the client-side some application control that is currently conducted at the server-side. In this case,

```
<xleaflet:RULE granularity="for_each_event" enabled="yes">
   <xleaflet:ON when="after"
               source="xleaflet:LINK[@to='/CONFERENCE/ACCEPTED_PAPER']"/>
   <xleaflet:IF>
      <xleaflet:TRUE/>
   </xleaflet:IF>
   <xleaflet:DO>
      <xleaflet:INSERT_TEMPLATE parents="/xleaflet:stylesheet">
         <xleaflet:template matchType="CONTENT"
                            matchInstance="/CONFERENCE/ACCEPTED_PAPER
                       [AREA='$FUNCTION:getNodeValue(&quot;$TO/AREA&quot;)']">
            <DIV style="background:blue; color:white">
               <b><xleaflet:value_of select="$matchInstance/TITLE"/></b><br/>
               <b>AUTHORS:</b><br/>
               <xleaflet:for_each select="$matchInstance/AUTHOR">
                  <xleaflet:value_of select="."/><br/>
               </xleaflet:for_each>
               <b>Electronic Edition: </b>
               <a target="newWindow">
                  <xsl:attribute name="href">
                  <xleaflet:value_of select="$matchInstance/ELECTRONIC_EDITION"/>
                  </xsl:attribute>
                  <xleaflet:value_of select="$matchInstance/ELECTRONIC_EDITION"/>
               </a>
               <br/>
               <b>AREA: <xleaflet:value_of select="$matchInstance/AREA"/></b>
               <br/>
               <b>DATE: <xleaflet:value_of select="$matchInstance/DATE"/></b>
               <br/>
               <b>ROOM: <xleaflet:value_of select="$matchInstance/ROOM"/></b>
               <br/> <b>TIME_SLOT:
                  <xleaflet:value_of select="$matchInstance/TIME_SLOT"/></b>
            </DIV>
         </xleaflet:template>
      </xleaflet:INSERT_TEMPLATE>
   </xleaflet:DO>
</xleaflet:RULE>
```

Figure 4.13: Adaptation browser rule: changing the presentation of related papers

the browser brought a chunk of data from where a whole bulk of pages could be locally generated with a single connection to the server. Navigation-intensive applications could be better served by this approach where the same content can be displayed several time without incurring in any communication cost.

This architecture implies that the *Xleaflet* is the unit of delivery. A *Xleaflet* expression is brought to the browser where a whole bulk of pages are generated *locally*. In this way, requests to the server are reduced to *(i)* retrieving the next *Xleaflet*, *(ii)* retrieving some resources (e.g. an imeage) or *(iii)* invoking a Web service which is attached to a derived element (see Chapter 2).

The Achilles' heel of this architecture is a more demanding browser configuration. So far, the *leaflet* run-time requires:

1. JavaScript libraries(or a plug-in) to interpret a *leaflet* document for navigation control and page generation.

2. A parser that processes the DOM API[Wea98], XSLT 1.0[Cla99] and XML Schema[TBMM01][5].

Next, we delve into the implementation details.

4.5.1 XLeaflet Architecture

The client-side *XLeaflet* architecture is shown in Figure 4.14[6]. An XLeaflet application is divided in two frames: namely, *LeafletVisualization* frame which supports the content display area, and *LeafletInterpreter* frame, which is hidden and where all run-time elements of *XLeaflet* are placed. More specifically, the *XLeaflet* run-time includes: (1) the *NavigationController* which maintains the current context and drives the navigation, (2) the *DocumentManager* module which is responsible for loading and updating all the *XLeaflet* documents, (3) the *PresentationController* which generates the HTML pages using the XSLT processor, (4) the *DeriveElementResolver* which invokes a web service in order to retrieve the external data, and (5) the *AdaptationController* which is in charge of adapting both navigation and presentation in accordance with the browser rules.

The initialization of an *XLeaflet* application is shown in Figure 4.15 using an interaction diagram. The browser requires an HTML initialization page to the WebServer which contains the four modules of the *XLeaflet* run-time. Then,

[5]This work uses Microsoft XML parser 4.0. So far, Internet Explorer 6.0 is the only browser that provides this DOM API and both XSLT and XML Schema processors. However, Netscape will also provide XML support in the next release.

[6]The diagram follows the UML notation proposed in [Con99]. This notation allows to model the implementation elements that compose a Web application by using UML stereotypes.

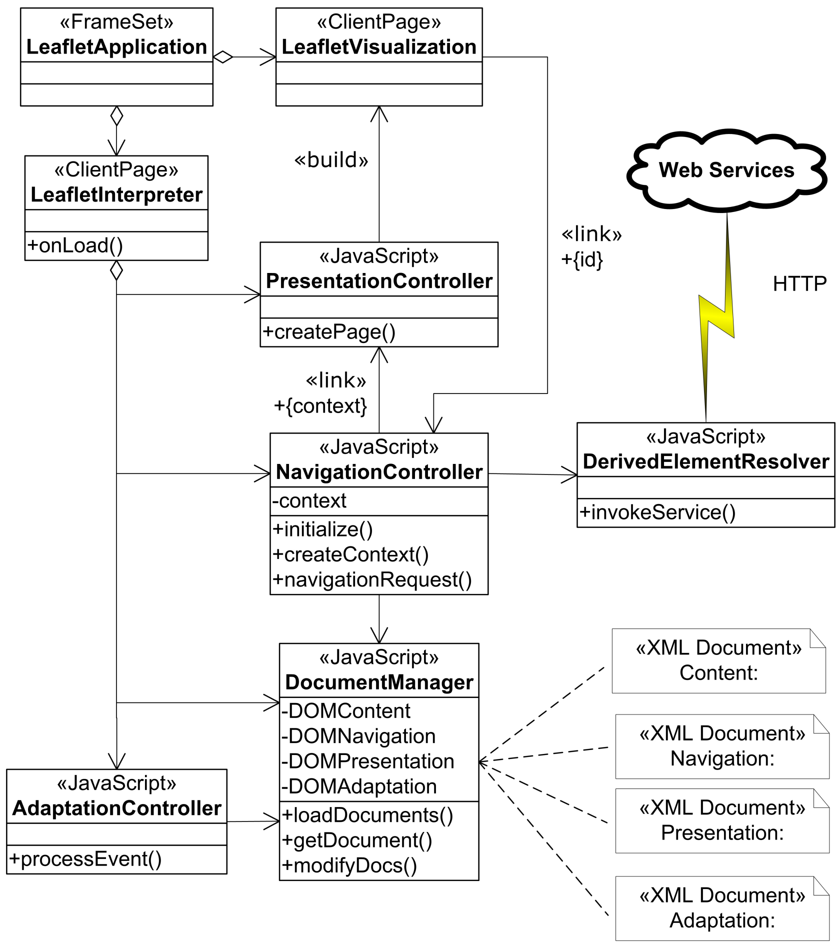


Figure 4.14: The XLeaflet architecture

the *initialize* method of the *NavigationController* is invoked, which renders the "home" page. To this end, the *NavigationController* requests the *DocumentManager* to load all the documents that constitute the *leaflet* (i.e. content, presentation, navigation and adaptation) and extracts from the navigation document the first links to traverse (i.e those having *"home"* as their `from` attribute, see section 4.4.2). At this time, the *AdaptationController* takes control over to fulfil the *procesEvent* request. The *AdaptationController* checks whether there is any "before" rule attached to the traversal of these home links; if so, the rule is triggered. If any of these rules is satisfied, as a result of rule´s action the content, presentation or navigation documents could be modified (by using the API provided by the *DocumentManager*). Next, the *NavigationController* undertakes link traversal, updates

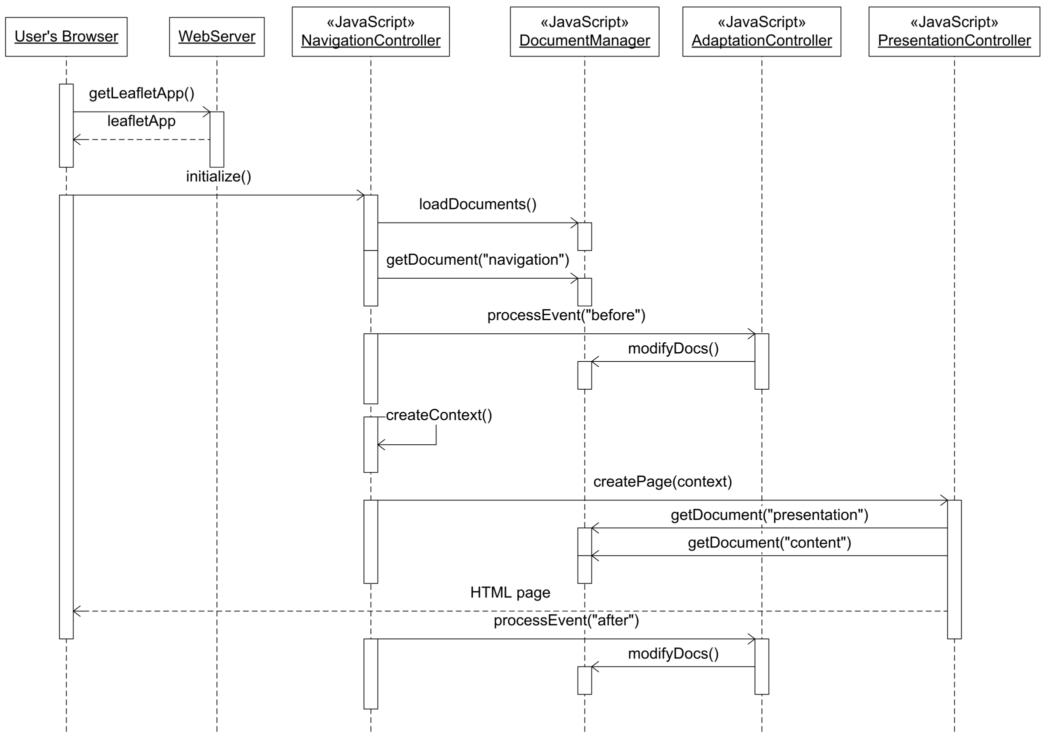


Figure 4.15: An interaction diagram for the XLeaflet initialization.

the current execution context, and requests to the *PresentationController* the rendering of the new page. To accomplish these tasks, the *PresentationController* requests the content and presentation documents to the *DocumentManager*. As a result, an HTML page is generated and rendered. Next, control returns to the *AdaptationController* to check for any "after" rule attached to those links just traversed. Once "after" rules have been processed, control goes back to the *NavigationController.* In case that the destination of a link traversal is an external data, then the NavigationController delegates to the *DeriveElementResolver* in order to retrieve it.

XLeaflet follows a light thick-web client architecture. The term "light" means that the configuration required for the client is kept to a minimum. So far, *XLeaflet* run-time is implemented using JavaScript libraries that interpret the different XML documents using the DOM API [Wea98], XSLT 1.0[Cla99] and XML Schema[TBMM01]. The widespread adoption of XML makes us feel confident about the support of this technology in most of the commercial browsers[7].

4.6 Evaluate Artefact

This section attempts to answer to what extent *XLeaflet* fulfils the requirements set in section 4.3: expressiveness and performance. The former has been accounted for several case studies:

- 6 websites in academic world: conference, 3D protein information, auction catalog, courses information, department information, and software catalog;
- 3 websites in business world: product catalog, questionaries-exams and factory-production information. The former requires updates to database on the server-side. This requirement was not initially consider on the design of the leaflet websites. However, this was solved defining a task (e.g. add, modify, remove) as another navigation mode in the navigation model. The task model and its implementation were maintained orthogonal and integrated with a transaction-oriented web-application [RD01], realized by J.J. Rodriguez in Onekin team.

The example use in this chapter (i.e. conference website) shows how common aspects that arise during *leaflet-website* development can be expressed through *XLeaflet* constructs. Other benefits brought by DSLs such as understandability or maintainability has not been addressed, though the higher abstraction terms in which the application is expressed is certainly a main rationale for achieving those goals.

As for performance, we next provide some figures about the cost of running a *XLeaflet* expression. On one hand, let's calculate the cost of rendering a traditional page. Three factors are involved:

- *(i)* a request for the page,
- *(ii)* the generation of the page at the server (if dynamically obtained),

[7]This work uses Microsoft XML parser 4.0. So far, Internet Explorer 6.0 is the only browser that provides this DOM API and both XSLT and XML Schema processors.

- *(iii)* the delivery of the page.

The former only accounts for less than one kilobyte and thus, it can be ignored. The other aspects are influenced by the bandwidth of the network, the size of the page and the amount of traffic both at the web server and the data server. An estimation for the required loading time can be obtained by using the following expression:

$$(size(Page)/bandwidth) + (size(Page)/Cs)$$

where *Cs* is a constant which reflects the server throughput (KB per second). We estimate that it takes 0.06 seconds for a web server to generate one kilobyte of HTML (of course, this is highly variable as it depends for instance on the load currently placed on the servers and the complexity of the SQL query). This gives a value for *Cs* of 15KB/sec.

On the other hand, rendering a *Xleaflet* involves: *(i)* a request for the *Xleaflet; (ii)* the delivery of the *Xleaflet* run-time; *(iii)* the delivery of the *Xleaflet* itself; and *(iv)* the processing of the *Xleaflet* at the browser.

If we ignore the first parameter, a possible formula reflecting these aspects is

$$\begin{aligned}(size(runtime)/bandwidth) &+\\ (size(Xleaflet)/bandwidth) &+\\ (size(Xleaflet)/Cb)&\end{aligned}$$

where *Cb* is a constant which reflects the *Xleaflet* run-time throughput (KB per second). We estimate that it takes 0.11 seconds for the run-time to generate one kilobyte of HTML. This gives a value for *Cb* of 9KB/sec.

However, this formula calculates the cost of processing a whole document from which several pages are generated. For the comparison to be done in equal terms, the previous formula should be expressed in terms of pages:

$$\begin{aligned}(size(runtime)/bandwidth) &+\\ (size(Xleaflet)/bandwidth) &+\\ \sum(size(GeneratedPage_i)/Cb)&\end{aligned}$$

Table 4.1: Captured data for evaluation: size and processing cost for each generated page for the conference example.

page size (KB)	processing cost(sec.)
10,38	0,59
5,66	0,81
7,25	0,67
8,78	0,96
8,77	1,14
8,80	0,96
8,73	0,95
8,76	0,95
7,44	0,70
8,02	0,76
8,91	1,17
8,92	1,18
8,91	1,08
9,20	1,26
10,15	1,42
10,73	1,59

Notice that the cost between generating one or several pages only differs on the third term of the formula. In other words, for the *Xleaflet* to be worth processing at the browser, several pages should be rendered. Otherwise, the cost of bringing both the run-time and the *Xleaflet* to the browser does not pay off. This is the reason why a thick-browser architecture only makes sense for navigation-intensive applications where a whole bulk of content need to be traversed in distinct ways. Furthermore, the data-intensive is at the very heart of the *Xleaflet* notion: a data assembly that makes sense to be browsed as a unit.

The question is what is the minimum number of pages to be rendered for the thick-browser architecture to payoff. In other words what is the value of *'numPag'* in:

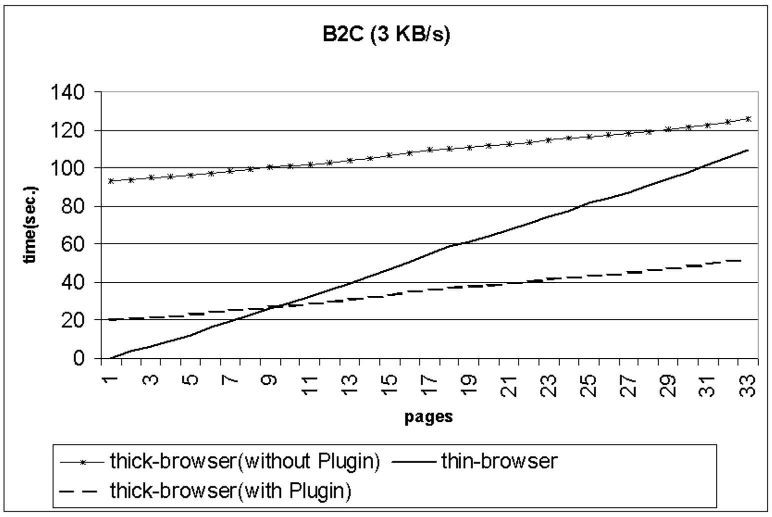


Figure 4.16: Evaluation: Time figures for a bandwidth of 3Kb/sec.

$$
\begin{aligned}
&(size(runtime)/bandwidth)+\\
&(size(Xleaflet)/bandwidth)+\\
&numPag\star(avg(size(Page_i))/Cb)\doteq\\
&numPag\star[(avg(size(Page))/bandwidth)+\\
&(avg(size(Page))/Cs)]
\end{aligned}
$$

For example, the conference *Xleaflet* has a size of 60K. This includes the content document, the navigation document and the presentation document. Consider the conference *Xleaflet* generates 32 pages; the sizes and the processing costs for the first sixteen pages are shown in table 4.1. As for the run-time, its current size is 220K.

Consider now three possible scenarios:

1. a thin-browser architecture where the *Xleaflet* is processed at the server,

2. a thick-browser architecture with the run-time is already installed as a plug-in, and the *Xleaflet* is processed at the browser

3. a thick-browser architecture where both the run-time and the *Xleaflet* are

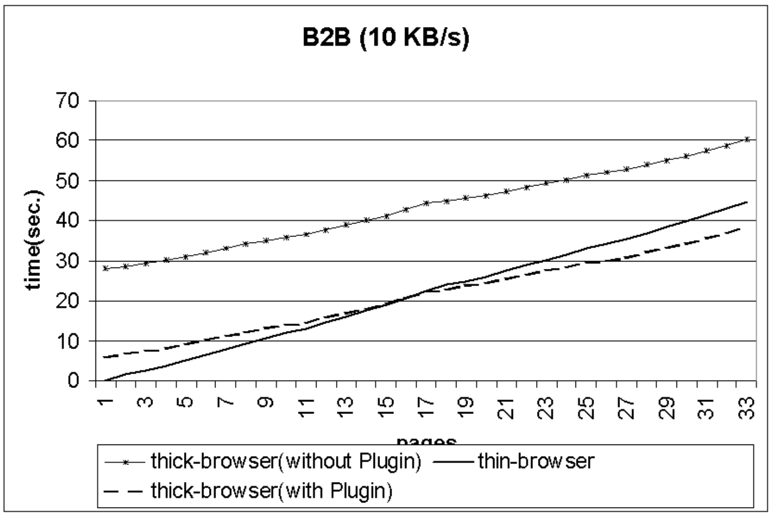


Figure 4.17: Evaluation: Time figures for a bandwidth of 10Kb/sec.

brought to the browser

Figure 4.16 and 4.17 compares this three options with a bandwidth of 3Kb/sec and 10Kb/sec, respectively.

4.6.1 Discussion

Although the *XLeaflet* run-time has not yet been tuned for efficiency optimization, we can already provide some insights from the previous figures:

First. The size of the run-time can make the thick-browser architecture inviable unless the run-time can be installed as a plug-in. This download penalty is payed just once as the plug-in caches the run-time the first time it is downloaded. Hence, this architecture does not fit B2C applications where users can be quite sporadic and could be discouraged by the perspective of having to install a plug-in. However, B2B applications offer a more promising setting. Firstly, the users are more computing-aware, and installing a plug-in will not put them off. Secondly, B2B relationships tend to be more stable that B2C relationships and thus, a business partner would be more willing to install the plug-in if better efficiency can be obtained while accessing one of its partner sites in the future. Another option is to provide a kind of run-time-lite which reduces the size of the system at the expense of reducing its functionality.

Second. The thick-browser architecture only pays off for navigation-intensive applications. As shown in Figure 4.16, this architecture begins to be faster than the traditional thin-browser architecture if 10 or more pages are rendered. For less than 10 pages, the download cost of the whole document does not pay off.

Third. The worse the network, the more interesting is the thick-browser architecture. When comparing Figures 4.16 and 4.17, it can be concluded that the minimum number of pages have to be rendered for the thick-browser architecture to pay off decreases as the bandwidth deteriorates.

4.7 Related work

Separation of concerns. A leaflet website development is similar to hypermedia and Web applications development. Most of the existing hypermedia methods, such as HDM[GPS93], RMM[ISB95], OOHDM [SR98], WSDM [TL98], Web-Composition [GWG97], WCML[GSG00], WebML[CFB00], AutoWeb[FP00], UWE[Koc01], OO-H [GC03], deal with the conceptual design of different concerns, such as content, navigation, presentation and user interaction. The main difference of them with this work is the architecture implementation. All of them are server-side approaches (i.e. thin-browser), whereas this work promotes a client-side approach (i.e. thick-browser).

4.8 Conclusions

This chapter focuses on so-called *leaflet websites*, addressing its development through a DSL. The outcome is *XLeaflet*, a DSL that strives to account for the expressiveness needs of these developments. In addition, *XLeaflet* also attempt to gain some insights into how to balance the application load between the browser and the server. So far, *XLeaflet* follows a thick-browser architecture where the navigation, presentation and adaptation controllers have been moved to the client-side. This is aligned with current XML technology where some browsers have been already enhanced to process XML documents. Moreover, this approach accounts for a reduction in the network traffic, an enhancement on the site's promptness, and a reduction of the server load. However, a downside of this approach is an increase of the demands on the browser configuration. This situation probably prevents the *Xleaflet* notion from being used for e-commerce applications targeted at final customers. However, B2B or e-procurement projects where more control over the client configuration is possible, could benefit from a thick-browser architecture.

Parts of the work described in this chapter have been previously presented:

- F. Ibáñez, O. Díaz, and J. J. Rodríguez. Coarse-grained delivery units: from HTML pages to XML Leaflets. In *Proceedings of the Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 311–318, Madrid, 2002.

- O. Díaz, F. Ibáñez, and J. Iturrioz. A model-based approach to web-application development. In *Semantic Issues in e-commerce systems (IFIP TC2 / WG2.6 Ninth Working Conference on Database Semantics)*, volume 111 of IFIP, pages 295–309, Hong Kong, April 25–28 2001. Kluwer Academic Publishers.

- O. Díaz, J. Iturrioz, and F. Ibáñez. Integración, navegación, presentación: experiencias utilizando XML. *Novatica,* (146):12– 19, 2000.

Chapter 5

Conclusions

5.1 Overview

The Extensible Markup Language (XML) provides a foundation for creating documents and document systems. XML is an opportunity to remake the software development landscape as well as the world of documents. XML has proven to be an adequate choice, as the file format, for both document and data exchange as well as Web development, and presents a new world of opportunities and challenges to programmers. However, from the use of XML in non-conventional applications emerge limitations to the current tooling and standards on XML. This dissertation proposes different solutions in order to overcome these limitations considering different scenarios. Mainly, a document-centric approach is promoted for both the development and the delivery of the application logic using XML.

This chapter reviews the main results of our work, evaluates the limitations and suggests work for future research.

5.2 Results

This dissertation develops the research content into three main chapters: Chapter 2 presented a model for extending an XML Schema in order to support the "missing data" (i.e. derived data) in an XML document; Chapter 3 faced how to define, compose and validate *XML deltas* in order to use XML documents in a *Software Product Line Engineering* setting that follows *Feature Oriented Software Development (FOSD)* paradigm; and Chapter 4 proposed thick-browser approach for web delivery content and separation of concerns for the development of a web application. Specifically,

- Chapter 2 presented a model for extending an XML Schema with **derived elements**, which will hold **derived data** (i.e. data which is calculated from other data) in an XML document. To this end, two issues had been discussed: deriving function specification and the externality of deriving data. The former had been addressed by introducing the *XDerive* vocabulary, a rule-based approach to the definition of the deriving function. It promotes code reuse, and saves the trouble of implementing these constraints in each application. As for the problems posed by the externality of deriving data, this chapter proposed the existence of a track document which records the state and business policies used at the time the transaction occurs. The result is a twofold XDerive Tool Suite: *(i)* a "derivation aware" XML parser and *(ii)* a derived data Explanator. The former extends an existing XML parser with a JAXP architecture. In this way, the schema is not only responsible for validating the instance document, but it is also in charge of generating the derived values by applying the corresponding business policies. The latter is a tool that explains the derivation process, revealing the deriving facts and the applied business policies, in order to enhance customer trustworthiness. In so doing, this approach eases the realisation of the distinct *contract* vocabularies currently emerging [GLC99], that focus on capturing the discovery-negotiation-execution life-cycle that models a business transaction.

- Chapter 3 introduced the notion of *XML delta* as a means of feature variability realisation in FOSD. The result is *XAK*, a standalone as well as an integrated tool in AHEAD Tool Suite. XAK allows to address the unit of composition in both an XML vocabulary and instance document, as well as define XML deltas which can be composed with a based document to output enhanced XML documents. Several different use cases were used to evaluate the approach. Moreover, XAK provides a validation of an XML delta with respect to an XML schema S in order to early detect inconsistencies that will synthesise an invalid documents with respect to S, provided the base document is also valid in S. Thus, a FOSD programmer can validate XML deltas before composition, at delta-definition time in an isolate way.

- Chapter 4 explored a new way of web content delivery in the era of XML. The result is *XLeaflet*, a model-based tool for web application development that renders HTML pages from the declarative schemata specified by the designer. Each concern (i.e. content, navigation, presentation and adaptation) is separately described in an XML document. A distinctive feature of *XLeaflet* interpreter is its architecture, i.e. client-intensive, the browser is in charge of *xleaflet* processing which involves managing both navigation control and page rendering. Thus, the advantages are twofold: enhancement of the web promptness and reduction in the server load.

5.3 Future Work

The three basic ideas explained so far has been implemented and proved as feasible using some sample cases. During the development of the solution some limitations were detected. Such limitations mark the direction of future work.

Enhancing business documents with derived data

- *Derived data maintenance*. Derived data maintenance is a fundamental problem in computer science. One of the issues of the derivation process is *when* deriving functions are executed. The XDerive parser follows a “snapshot approach”, both the deriving facts and the derived data are obtained and calculated at parsing time of a business document. However, the external data used by deriving functions can be changed in the database and make invalid the derived data. This poses the challenge of analysing more business cases in order to balance between the importance of data freshness and transaction response time. The last include the study of an implementation of a distributed trigger system, that integrates an XML parser and a database system.

On refining XML artefacts

- *Domain-specific delta composition.* Currently, our definition of XML deltas only permits additions and updates of existing module elements. The XAK composer implements a generic XML composition and a delta XML Schema is derived from the domain XML Schema based on the generic composition. However, generic composition is not sufficient in certain cases and a domain-specific composition need to be defined [Ses11]. Studying in which cases specific composition is required, what are its drawbacks and the feasibility to use our XDerive approach is an area for future work.

- *Safe Composition of Documents.* When developing a family of documents incrementally, we want to guarantee that all legal feature compositions (i.e., all feature compositions that correspond to a product) yield documents that satisfy all constraints of the XML Schema, *but without* generating and validating one by one. Our work paves the way to this goal by allowing to check delta constraints. Nevertheless, the question of how to check the compatible input documents for each delta document to guarantee that every product conforms to its XML Schema still remains open. Existing work using propositional formulas and SAT-solvers suggests a direction in which to proceed [DCB09, AKGL10].

- *Semantic constraint validation.* Generally, XML validation can be classified into two orientations: structural constraint validation and semantic constraint validation (e.g keys/ID, functional dependencies) [WP11]. The validation of a delta document before composition is a main property for safe document composition in FOSD. Our work validates a delta document structurally, but not semantically. It can be verified after composition. However, there are few work in this area and a complete check can be too costly. Hence, a more efficient solution for the verification of semantic constraints requires to derive also specialised constraints in order to detect illegal delta documents before composition. Hence, the validation will be complete. This type of validation will be important in the area of FOSD that includes models (e.g. XMI), XML process definitions or business documents.

- *Variability of XML Schema definition.* An XML schema is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type. In our work, the schemata is fixed and the variability part implemented by features is in the content described by documents, but more complex applications may demand to extend the vocabulary described by a schema, e.g. extension of content models with new elements or attributes, alternatives of content models. For example, services in a Service-Oriented Architecture are described by WSDL, which describes operations and includes XML schemas for exchanging the structure of XML documents among different components of a system. Thus, a feature that adds more operations in a service may require to define new types or elements, or extend existing ones in the XML Schema definition part. In this context, more work is needed to evaluate the suitability of the XAK approach.

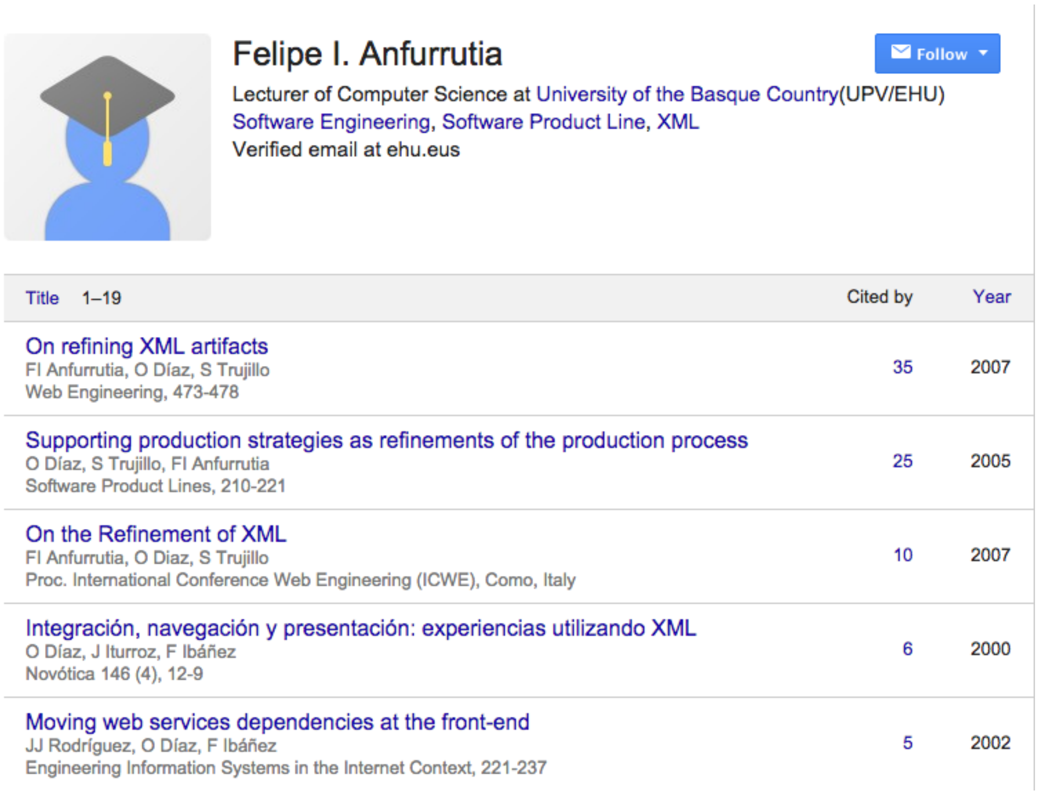
The XLeaflet approach

- *Hybrid architecture*. XLeaflet proposes a thick-browser (i.e. client-side) architecture instead of a thin one (i.e. server-side). However, it is conceivable to apply these two different patterns (i.e. the thick and the thin pattern) to a single hybrid architecture, where both the thick- and the thin-architecture approaches can coexist in the very same application. The degree of control of the client's configuration and the data-load of each use-case supported by the application can dictate whether a think or a thin pattern is more convenient. The application could be split into a set of *leaflet*s. Where the *leaflet* will be processed (i.e. the server or the browser) depending on the expected navigation pattern and how much data will be derived from external sources. “Navigation-intensive” *leaflet*s will be moved to the browser while “data-intensive” *leaflet*s are preferably kept at the server.

- *Operations related to content.* Web applications are not only to show content but also to execute operations. These operations can be related to the shown content. Hence, there is another concern related to the content model. Previous attempt describes the operations as part of the navigation model as the operation is another form to traverse the content document. However, more work is needed to determine what are the primitive operations offer by the framework and whether the operations will be model independently or as part of the navigation model.

- *Reactive rules related to visited content.* Reactive rules are used for programming rule-based, reactive systems, which have the ability to detect events and respond to them automatically in a timely manner. Such systems are needed on the Web for bridging the gap between the existing, passive Web, where data sources can only be accessed to obtain information, and the dynamicWeb, where data sources are enriched with reactive behaviour. Differences between (generally centralised) active databases and the Web, where a central clock, a central management are missing and new data formats (such as XML) are used, give reasons for developing new approaches based on different kinds of reactive rules, namely Event-Condition-Action rules and production rules.

- *MDD approach.* Model Driven Development and Generative Programming. Model-driven engineering coupled with code generation can provide enormous benefits in terms of developers’ productivity, reduced development and maintenance costs, shorter time-to-market, and improved product quality. Using XML technologies for modeling, validations, transformations, and code generation proves to be a pretty low-cost and yet extremely powerful approach to model driven development.



Felipe I. Anfurrutia

Lecturer of Computer Science at University of the Basque Country(UPV/EHU)
Software Engineering, Software Product Line, XML
Verified email at ehu.eus

Title 1–19	Cited by	Year
On refining XML artifacts FI Anfurrutia, O Díaz, S Trujillo Web Engineering, 473-478	35	2007
Supporting production strategies as refinements of the production process O Díaz, S Trujillo, FI Anfurrutia Software Product Lines, 210-221	25	2005
On the Refinement of XML FI Anfurrutia, O Diaz, S Trujillo Proc. International Conference Web Engineering (ICWE), Como, Italy	10	2007
Integración, navegación y presentación: experiencias utilizando XML O Díaz, J Iturroz, F Ibáñez Novótica 146 (4), 12-9	6	2000
Moving web services dependencies at the front-end JJ Rodríguez, O Díaz, F Ibáñez Engineering Information Systems in the Internet Context, 221-237	5	2002

Figure 5.1: Top five publications as for the number of references in Google Scholar [Accessed 8 December 2015].

5.4 Publications

Parts of the work explained in this thesis have been presented and discussed in distinct peer-reviewed forums. Figure 5.1 depicts the top five publications as for the number of references in Google Scholar. A more detailed account follows.

Selected Publications

- *On Refining XML Artifacts*. F. I. Anfurrutia, O. Díaz and S. Trujillo. In Proceedings of the 7th International Conference on Web Engineering (ICWE'07), Como (Italy), July 2007 [ADT07]. Rank **B** in the ERA Conference Ranking. Acceptance Rate: 23% (26+13/172)

- *Supporting Production Strategies as Refinements of the Production Process*. O. Díaz, S. Trujillo, F. I. Anfurrutia. In Proceedings of the Software Product

Line Conference (SPLC'05), Rennes (France), September 2005 [DTA05]. Acceptance Rate: 23% (17+3/71).

- *Integración, navegación, presentación: experiencias utilizando XML*. O. Díaz, J. Iturrioz and F. Ibañez. Novatica, no. 146, pp. 12-19, 2000[DII00].

- *Using DITA for documenting Software Product Lines*. O. Díaz, F. I. Anfurrutia and J. Kortabitarte. Proceedings of the 9th ACM symposium on Document engineering (DocEng'09), 231-240. September 2009 [DAK09]. Rank **A** in the ERA Conference Ranking. Acceptance rate: 31%.

- *Improving self-interpretation of XML-based business documents by introducing derived elements*. O. Díaz and F. I. Anfurrutia. Electronic Commerce Research and Applications (ECRA), vol. 4, pp. 264-282, May 2005[DA05]. **JCR**, SJR Impact factor: 0.657, Q2

International Conferences

- *Coarse-grained delivery units: from HTML pages to XML Leaflets*. F. Ibáñez, O. Díaz and J. J. Rodríguez. In Proceedings of the Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'02), pp. 311-318, 2002[IDR02a]. Rank **C** in the ERA Conference Ranking.

- *Extending XML Schema with Derived Elements*. F. Ibáñez, O. Díaz and J. J. Rodríguez. In Proceedings of the IFIP TC8 / WG8.1 Working Conference on Engineering Information Systems in the Internet Context , Kluwer Academic Publishers, vol. 231, pp. 53-67, 2002[IDR02b].

- *A tool for assessing the consistency of Websites*. S. Steinau, O. Díaz, J. J. Rodríguez, and F. Ibáñez. In Proceedings of the 4th International Conference on Enterprise Information Systems, vol. 2, pp. 691-698, 2002[SDRI02a]. Rank **B** in the ERA Conference Ranking. Acceptance rate: 40% (89/220). This paper was among best conference papers (it was later published in a book version [SDRI02b]). Acceptance rate: 13% (30/220)

- *Wrapping HTML pages as Interactive Web Services*. O. Díaz, Juan J. Rodríguez, I. Paz and F. Ibáñez. In Proceedings of the Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'02), pp. 301-309, 2002[DRPI02]. Rank **C** in the ERA Conference Ranking.

- *Moving Web Services Dependencies at the front-end*. J. J. Rodriguez, O. Diaz and F. Ibáñez. In Proceedings of the IFIP TC8 / WG8.1 Working Conference on Engineering Information Systems in the Internet Context, Kluwer Academic Publishers, vol. 231, pp. 221-237, 2002[RDA02].

- *An Overview on XML initiatives to Bring Modularization to Web Application Development*. O. Díaz, A. Irastorza, J. J. Rodríguez and F. I. Anfurrutia. In Proceedings of the WWW/Internet 2002 (IADIS International Conference), pp. 435-443, 2002[DIRA02]. Rank **C** in the ERA Conference Ranking.

- *A model-based approach to web-application development*. O. Díaz, F. Ibáñez and J. Iturrioz. In *Semantic Issues in e-commerce systems (IFIP TC2 / WG2.6 Ninth Working Conference on Database Semantics)*, volume 111 of IFIP, pages 295–309, Hong Kong, April 25–28 2001. Kluwer Academic Publishers [DII01b].

National Journal/Conferences/Workshops

- *A client intensive, a model-based approach to web application development: The AtariX system*. O. Díaz, F. Ibáñez and J. Iturrioz. In Workshop on Ingeniería del Software orientada a la Web (co-located with JISBD'01), Almagro, Ciudad Real (Spain), 2001[DII01a].

- *XDerive: a namespace for defining derived elements in XML Schema*. O. Díaz and F. I. Anfurrutia. In Workshop on Métodos y Herramientas para el comercio electrónico (ZOCO'02, co-located with JISBD'02), El Escorial, Madrid (Spain), 2002[DA02].

- *A Product-Line Approach to Database Reporting*. F. I. Anfurrutia, O. Diaz and S. Trujillo. In Proceedings of the X Jornadas sobre Ingeniería del Soft-

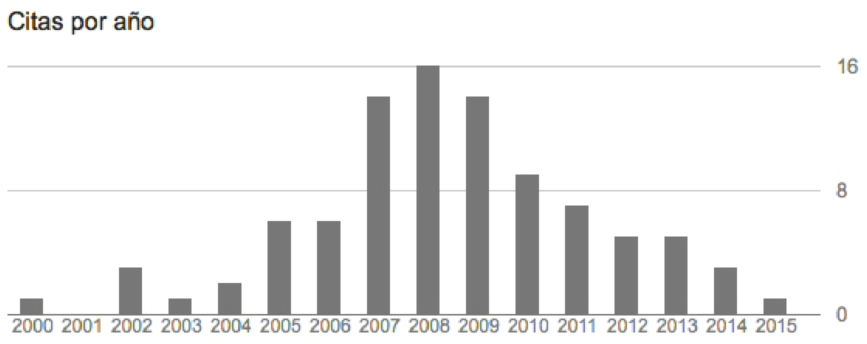


Figure 5.2: Cites per year obtained from https://scholar.google.es/citations?user=lc_158gAAAAJ&hl=es

ware y Bases de Datos (JISBD'05), 2005, Granada (Spain) [ADT05]. Acceptance Rate: 31 % (29+10/92). This paper was among best 10 conference papers (it was later published in a journal version [ADT06]).

- *Una Aproximación de Línea de Producto para la Generación de Informes de Bases de Datos* (Spanish version of [ADT05]). F. I. Anfurrutia, O. Diaz and S. Trujillo. IEEE América Latina Journal (IEEE-AL), vol. 4, no. 2, April 2006 [ADT06]. **Impact factor: 0.218** (IEEE Xplore)

Figure 5.2 summarises cites per year as an indicator of the impact that our work has in the community. **The total number of citation is 93**. Next, the "cited by" list of publications is presented (obtained from Scopus and Google Scholar), where self citations are removed.

"cited by"

- [ADT07] is cited by 13 articles:
  - Feature-oriented software development a short tutorial on feature-oriented programming, virtual separation of concerns, and variability-aware analysis. Kästner, C., Apel, S. LNCS. 2013
  - Frame refinement: combining frame-based software development with stepwise refinement Authors of Document. Zhou, J., Zhao, D., Xu, L., Liu, J. Computer Research and Development. 2013

– A design feature-based approach to deriving program code from features: A step towards feature-oriented software development. Lee, H., Kang, K.C. ACM International Conference Proceeding Series. 2013

– Do we need another textual language for feature modeling? A preliminary evaluation on the XML based approach. Zhou, J., Zhao, D., Xu, L., Liu, J. Studies in Computational Intelligence. 2012

– An algebraic foundation for automatic feature-based program synthesis. Apel, S., Lengauer, C., Möller, B., Kästner, C. Y. Science of Computer Programming. 2010

– Type safety for feature-oriented product lines. Apel, S., Kästner, C., Größlinger, A., Lengauer, C. Automated Software Engineering. 2010

– A calculus for uniform feature composition. Apel, S., Hutchins, D. ACM Transactions on Programming Languages and Systems. 2010

– FeatureHouse: Language-independent, automated software composition. Apel, S., Kästner, C., Lengauer, C. Proceedings - International Conference on Software Engineering. 2009

– Feature-oriented refinement of models, metamodels and model transformations. Trujillo, S., Zubizarreta, A., Mendialdua, X., De Sosa, J. ACM International Conference Proceeding. 2009

– An orthogonal access modifier model for feature-oriented programming. Apel, S., Liebig, J., Kästner, C., Kuhlemann, M., Leich, T. ACM International Conference Proceeding Series. 2009

– An overview of feature-oriented software development. Apel, S., Kästner, C. Journal of Object Technology. 2009

– Research challenges in the tension between features and services. Apel, S., Kästner, C., Lengauer, C. Proceedings - International Conference on Software Engineering. 2008

– Feature featherweight java: A calculus for feature-oriented programming and stepwise refinement. Apel, S., Kästner, C., Lengauer, C. GPCE'08: Proceedings of the ACM SIGPLAN 7th International Conference on Generative Programming and Component Engineering. 2008

- [ADT06] is cited by 2:
  - CoDe modeling of graph composition for data warehouse report visualization. Risi, M., Sessa, M.I., Tucci, M., Tortora, G. IEEE Transactions on Knowledge and Data Engineering. 2014
  - Visualizing information in data warehouses reports. Risi, M., Sessa, M.I., Tortora, G., Tucci, M. SEBD 2011, Proceedings of the 19th Italian Symposium on Advanced Database Systems. 2011
- [DAK09] is cited by 4:
  - Flexible support for managing evolving software product lines. Thao, C., Munson, E.V. Proceedings - International Conference on Software Engineering. 2011
  - Challenges to Establish Internal Quality Assurance with An Information System to Create Self-assurance Report. M Mori, E Takata, T Oishi, T Tanaka. New Perspectives in Science Education. 2014
  - Documentation Agile: Pratiques Actuelles et Défis. A Hachemi, M Ahmed-Nacer - CIIA. 2011
  - A Document Authoring System for Credible Enterprise Reporting with Data Analysis from Data Warehouse. M Mori, T Tanaka, S Hirokawa. SEMAPRO 2010, The Fourth International Conference on Advances in Semantic Processing. 2010

5.5 About Design Science

Back in the introduction, Design Science was defined as “the scientific study and creation of artefacts as they are developed and used by people with the goal of solving practical problems of general interest”. We would like to highlight two aspects from this definition: the pivotal role played by artefacts, and the solution of practical problems as the driving force that move forward artefact development. This definition seems to imply that without artefacts or problems, no Design Science exist.

In Design Science, artefacts are not just a second thought but the cornerstone around which the research resolves. Moving the artefact at the forefront, it is certainly not a surprise for engineers but its importance has sometimes being overlooked in academia which tends to look at them as mere programming exercises. As a counterbalance, artefacts do not exist in isolation. Artefacts' raision d'etre come from problems. Artefacts exist as long as problems exist. The value of an artefact comes from the problem it solves. An artefact without a problem is just a programming caper.

These thoughts might look obvious... now. But they were not at the time this thesis started. Though this dissertation strives to follow Design Science principles, this was not the case at its inception. Ten years back, we were not so familiarized with Design Science. The result is that this work is biased towards the artefact while not given enough attention to the problem. Enormous amount of times was dedicated to make things work while not equal treatment was given to conduct cause root analysis or evaluation. Albert Einstein once said, "If I had a hour to solve a problem, I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions". I have to admit I was not Albert Einstein! Sharing these thoughts with my supervisor, Oscar, he also admits that there exists a natural tendency for Software Engineering students to quickly delve into the code without carefully thinking about the requirements that should drive this development. Students tend to favour "building the software right" rather than "building the right software". Programming is fun, analysis is bouring. Research wise, this bias is unfortunate. I keep reading MSc dissertations or even conference papers where the rationales are unclear, or even worse, can be easily denied since a proper root cause analysis was not conducted. This bias might be traced back to Soft Eng. syllabuses that, overwhelmed by technological advances, provide limited contents about domain analysis and software evaluation. The outcome is that students enter industry striving to "building the sofware right", what is good, but get frustrated when customers keep saying that they want "the right software"!

Should I had (even) more time, what would I do? Without any doubt, I would focus on improving evaluation of the different artefacts developed throughout. This is the weakest part of this dissertation. It somehow reflects the stage-of-the-affairs at the time this thesis was initiated. Of course, the larger weight given to evaluation should not be at the expenses of development and technical competence. This would be an error. For PhD students, technical competence is in most cases the entry door to industry. Industry looks for people building the system right. Even if you are great at requirement elicitation and domain analysis, chances are that what industry will look for in your CV are your programming language mastering. And this is the dilemma. While Design Science favours a holistic view that encompasses from analysis to evaluation, the competences required throughout are difficult to achieve in the shrinking lifespan of a PhD (three years). This in itself is a problem worth addressing using Design Science itself!! Talking with my supervisor, he envisions two ways out. First, doctoral programs need to give more focus to the methodology of research rather than to technical/scientific contents. Second, team work. PhDs should no longer be a secluded activity but inlayed within a group. Unfortunately, Design Science courses, courses about artefact evaluation or teamwork seem all to be the exception rather than the norm. If this continues, we will keep seeing poor PhDs that expand over three years.What a pity!

Appendix A

The specification of XDerive namespace

```
<xs:schema xmlns="http://www.onekin.org/xderive"
           targetNamespace="http://www.onekin.org/xderive"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           elementFormDefault="qualified">
  <xs:annotation>
    <xs:documentation>This schema is used as a part of another
        schema, which defines an XML vocabulary that contains
        derived data.</xs:documentation>
  </xs:annotation>
  <xs:import namespace="http://www.w3.org/1999/XSL/Transform"
             schemaLocation="xslt.xsd"/>
  <xs:element name="derivingFunction" type="DerivingFunctionType">
    <xs:annotation>
      <xs:documentation>Purpose:  This element defines the rules for
             calculating the value of derived data.
          Context:  It is used as a child of &lt;xs:appinfo> element
             within a derived element or attribute declaration
          Constraint:  minOccurs of derived element or attribute should be 0
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="DerivingFunctionType">
    <xs:sequence>
      <xs:element name="documentation" type="xs:string" minOccurs="0"/>
      <xs:element name="rule" type="RuleType" maxOccurs="unbounded"/>
      <xs:element name="prioritizationSchema" type="PrioritizationSchemaType"
                  minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="actuate" use="optional" default="onLoad">
      <xs:simpleType> <xs:restriction base="xs:string">
        <xs:enumeration value="onLoad"/>
        <xs:enumeration value="onRequest"/>
      </xs:restriction> </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="RuleType">
    <xs:sequence>
      <xs:element name="documentation" type="xs:string" minOccurs="0"/>
      <xs:element name="action" type="ActionType"/>
      <xs:element name="source" type="SourceType" minOccurs="0"
                  maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:NMTOKEN" use="required"/>
    <xs:attribute name="test" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="ActionType">
    <xs:choice>
      <xs:sequence>
        <xs:element ref="xsl:instruction" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:element name="apply-rule" type="ApplyRuleType"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="ApplyRuleType">
    <xs:attribute name="select" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="SourceType">
    <xs:sequence>
      <xs:any namespace="http://www.oracle.com/xsql"/>
    </xs:sequence>
    <xs:attribute name="derivingFact" type="xs:NMTOKEN" use="required"/>
    <xs:attribute name="connection" type="xs:NMTOKEN" use="required"/>
  </xs:complexType>
  <xs:complexType name="PrioritizationSchemaType">
    <xs:choice>
      <xs:element name="valueBased" type="ValueBasedType"/>
      <xs:element name="ruleBased" type="RuleBasedType"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="ValueBasedType">
    <xs:attribute name="combiningFunction" type="xs:string"
                  use="required"/>
  </xs:complexType>
  <xs:complexType name="RuleBasedType">
    <xs:sequence>
      <xs:element name="documentation" type="xs:string" minOccurs="0"/>
      <xs:element name="rule" type="RuleReducedType"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="RuleReducedType">
    <xs:complexContent>
      <xs:restriction base="RuleType">
        <xs:sequence>
          <xs:element name="documentation" type="xs:string"
                      minOccurs="0"/>
          <xs:element name="action" type="ActionType"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

</xs:schema>
```

Figure A.1: The W3C XML Schema representation for *XDerive* namespace

Appendix B

XDerive tracking document

```
<?xderive-source href="order.xml" type="text/xml"?>
<xdt:derivationTrack xmlns:xdt="http://www.onekin.org/xderive/derivationTrack"

  <xdt:derivingElements>

    <category ref="TV-1">electronic</category>
    <category ref="Hi-Fi-1">electronic</category>
    <category ref="PC-1">computers</category>
    <availability ref="TV-1">24 hours</availability>
    <availability ref="Hi-Fi-1">24 hours</availability>
    <availability ref="PC-1">24 hours</availability>
    <specialHanging ref="TV-1">false</specialHanging>
    <specialHanging ref="Hi-Fi-1">false</specialHanging>
    <specialHanging ref="PC-1">true</specialHanging>
    <freeShipping ref="TV-1">true</freeShipping>
    <freeShipping ref="Hi-Fi-1">true</freeShipping>
    <freeShipping ref="PC-1">true</freeShipping>
    <unitPrice ref="TV-1">1000.00</unitPrice>
    <unitPrice ref="Hi-Fi-1">600.00</unitPrice>
    <unitPrice ref="PC-1">1200.00</unitPrice>
  </xdt:derivingElements>
  <xdt:derivedElements>
    <xdt:derivedElement select="/order/lineItem[1]/partialCost">
      <xdt:activated rules="rule-20"/>
      <xdt:decision type="priority rule" choice="no"/>
      <xdt:executed rule="rule-20"/>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/lineItem[2]/partialCost">
      <xdt:activated rules="rule-20"/>
      <xdt:decision type="priority rule" choice="no"/>
      <xdt:executed rule="rule-20"/>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/lineItem[3]/partialCost">
      <xdt:activated rules="rule-20"/>
      <xdt:decision type="priority rule" choice="no"/>
      <xdt:executed rule="rule-20"/>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/shippingData/shipments">
      <xdt:activated rules="rule-5 rule-6"/>
      <xdt:decision type="priority rule" choice="1"/>
      <xdt:executed rule="rule-6"/>

    </xdt:derivedElement>
    <xdt:derivedElement select="/order/shippingData/insuranceCost">
      <xdt:activated rules="rule-9 rule-10"/>
      <xdt:decision type="combining function"/>
      <xdt:executed rule="rule-9">
        <xdt:actionResult>5.99</xdt:actionResult>
      </xdt:executed>
      <xdt:executed rule="rule-10">
        <xdt:actionResult>7.99</xdt:actionResult>
      </xdt:executed>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/shippingData/shippingCost">
      <xdt:activated rules="rule-14"/>
      <xdt:decision type="priority rule" choice="4"/>
      <xdt:executed rule="rule-14"/>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/shippingData/deliveryTime">
      <xdt:activated rules="rule-18"/>
      <xdt:decision type="priority rule" choice="3"/>
      <xdt:executed rule="rule-18"/>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/customer/customerData">
      <xdt:activated rules="rule-19"/>
      <xdt:decision type="priority rule" choice="no"/>
      <xdt:executed rule="rule-19"/>
      <xdt:faults>
        <xdt:fault name="cis:nonExistFault"/>
      </xdt:faults>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/applicableDiscount">
      <xdt:activated rules="rule-21 rule-22 rule-23"/>
      <xdt:decision type="combining function"/>
      <xdt:executed rule="rule-21">
        <xdt:actionResult>10</xdt:actionResult>
      </xdt:executed>
      <xdt:executed rule="rule-22">
        <xdt:actionResult>20</xdt:actionResult>
      </xdt:executed>
      <xdt:executed rule="rule-23">
        <xdt:actionResult>5</xdt:actionResult>
      </xdt:executed>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/subTotal">
      <xdt:activated rules="rule-1"/>
      <xdt:decision type="priority rule" choice="no"/>
      <xdt:executed rule="rule-1"/>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/vat">
      <xdt:activated rules="rule-3"/>
      <xdt:decision type="priority rule" choice="no"/>
      <xdt:executed rule="rule-3"/>
    </xdt:derivedElement>
    <xdt:derivedElement select="/order/total">
      <xdt:activated rules="rule-4"/>
      <xdt:decision type="priority rule" choice="no"/>
      <xdt:executed rule="rule-4"/>
    </xdt:derivedElement>
  </xdt:derivedElements>

</xdt:derivationTrack>
```

Appendix C

The delta grammar for the form-bean module type

Let's consider the Struts XML Schema and the *form-bean* element type that is annotated as a module type (i.e. *xak:modularizable="yes"*):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!–content omitted –>
  <xs:element name="form-bean" xak:modularizable="yes">
    <xs:complexType>
      <xs:complexContent>
        <xs:sequence>
          <xs:element name="description" type="xs:string" minOccurs="0"/>
          <xs:element name="form-property" minOccurs="0" maxOccurs="unbounded">
              <!–content omitted–>
          </xs:element>
        </xs:sequence>
      </xs:complexContent>
    </xs:complexType>
  <xs:element/>
<xs:schema/>
```

The previous module type can be represented by the following regular expression in XDuce syntax [HVP05]:

$$type\ T_{form-bean} = \mathbf{form} - \mathbf{bean}[description[xs : string]?, form - property[]*]$$

We can obtained $\triangle T_{form-bean}$ delta module type after applying the $\triangle moduleT$ function (see 3.5.3) to the previous $T_{form-bean}$ type. The type and its related types are as follows:

$$\begin{aligned}
type\ \triangle T_{form-bean} &= \mathbf{form-bean}[description[xs:string], \\
&\quad (xak:keep-content[], X_{form-property}?| \\
&\quad X_{form-property-kc})?|X_{form-property-kc}| \\
&\quad xak:keep-content[], X_{form-property}] \Rightarrow \\
\{\vdash typeX_{form-property} &= form-property[]+; \\
typeX_{form-property-kc} &= form-property[]+, xak:keep-content[]?, \\
&\quad X_{form-property}?\}
\end{aligned}$$

Finally, Figure C.1 shows the obtained delta module type $\triangle T_{form-bean}$, now represented by an XML Schema. All its related type names, ranged over by X and used in the previous definition, are transformed into a `<xs:group>` construct in order to reuse definitions. Notice that the `form-bean` element is included in the *substitutionGroup* of the abstract `<xak:deltaModule>` element and its type extends the `xak:DeltaModuleType`. The former is for permitting its use as a delta module inside the `<xak:refines>` element, i.e. the root of the delta document. Whereas the latter is for inheriting the `xak:module` and `xak:keep-attribute` attributes. Figure C.2 shows all these XAK elements and attributes definitions.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:xak="http://www.onekin.org/xak">
  <xs:import namespace="http://www.onekin.org/xak" schemaLocation="xak.xsd"/>
  <xs:element name="form-bean" type="form-beanType"
                        substitutionGroup="xak:deltaModule"/>
  <xs:complexType name="form-beanType">
    <xs:complexContent>
      <xs:extension base="xak:DeltaModuleType">
        <xs:choice minOccurs="0">
          <xs:group ref="X_description"/>
          <xs:group ref="X_form-property-kc"/>
          <xs:sequence>
            <xs:element ref="xak:keep-content"/>
            <xs:group ref="X_form-property"/>
          </xs:sequence>
        </xs:choice>
       <!–content omitted–>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:group name="X_description">
    <xs:sequence>
      <xs:element name="description" type="xs:string"/>
      <xs:choice minOccurs="0">
        <xs:sequence>
          <xs:element ref="xak:keep-content"/>
          <xs:element ref="X_form-property" minOccurs="0" />
        </xs:sequence>
        <xs:group ref="X_form-property-kc"/>
      </xs:choice>
    </xs:sequence>
  </xs:group>
  <xs:group name="X_form-property-kc">
    <xs:sequence>
      <xs:element name="form-property" maxOccurs="unbounded">
           <!–content omitted–>
      <xs:element/>
      <xs:sequence minOccurs="0">
        <xs:element ref="xak:keep-content"/>
        <xs:group ref="X_form-property" minOccurs="0" />
      </xs:sequence>
    </xs:sequence>
  </xs:group>
  <xs:group name="X_form-property">
    <xs:sequence>
      <xs:element name="form-property" maxOccurs="unbounded">
           <!–content omitted–>
      <xs:element/>
    </xs:sequence>
  </xs:group>
  <!–content omitted–>
</xs:schema>
```

Figure C.1: An excerpt of the obtained delta grammar for Struts, represented by XML-Schema, which defines the *form-bean* delta module type.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns:xak="http://www.onekin.org/xak"
                   targetNamespace="http://www.onekin.org/xak"
                   elementFormDefault="qualified" attributeFormDefault="qualified">
  <xs:element name="refines" type="xak:DeltaType" />
    <xs:complexType name="DeltaType">
      <xs:attribute name="artefact" type="xs:string" use="required"/>
      <xs:attribute name="feature" type="xs:string" use="required"/>
    </xs:complexType>
  <xs:element name="deltaModule" type="xak:DeltaModuleType" abstract="true"/>
  <xs:complexType name="DeltaModuleType">
    <xs:attribute ref="xak:module" use="required"/>
    <xs:attribute name="keep-attributes" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:attribute name="module" type="xs:ID"/>
  <xs:element name="keep-content"/>
</xs:schema>
```

Figure C.2: The XML Schema for XAK delta namespace

Bibliography

[ABKS13] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer, 2013.

[ABM$^+$04] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: A Data-Centric Perspective on Web Services. In *Web Dynamics*, pages 275 – 299. Springer Berlin Heidelberg, 2004.

[ADT05] F. I. Anfurrutia, O. Díaz, and S. Trujillo. A product-line approach to database reporting. In *Proceedings of the X Jornadas sobre Ingeniería del Software y Bases de Datos (JISBD'05)*, pages 163–170, Granada, Spain, 2005. Thompson.

[ADT06] F. I. Anfurrutia, O. Díaz, and S. Trujillo. Una Aproximación de Línea de Producto para la Generación de Informes de Bases de Datos (A product-line approach to database reporting). *IEEE América Latina*, 4, April 2006. ISSN: 1548-0992. Available online at http://www.ewh.ieee.org/reg/9/etrans/vol4issue2April2006/Vol4issue2April2006TLA.htm.

[ADT07] F. I. Anfurrutia, O. Díaz, and S. Trujillo. On refining XML artifacts. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, *Proceedings of the 7th International Conference on Web Engineering (ICWE'07)*, volume 4607 of *LNCS*, pages 473–478. Springer, 2007.

[AGMW97] B. Adelberg, H. Garcia-Molina, and J. Widom. The STRIP rule system for efficiently maintaining derived data. *SIGMOD Rec.*, 26(2):147–158, 1997.

[AKGL10] S. Apel, C. Kästner, A. Größlinger, and C. Lengauer. Type Safety for Feature-oriented Product Lines. *Automated Software Engineering*, 17(3):251–300, September 2010.

[AKL09] S. Apel, C. Kastner, and C. Lengauer. FEATUREHOUSE: Language-independent, automated software composition. In *Proceedings of the 31st International Conference on Software Engineering, 2009. ICSE 2009. IEEE*, pages 221–231, May 2009.

[alp01] IBM alphaWorks. XML Diff and Merge Tool, 2001. Available online at http://www.alphaworks.ibm.com/tech/xmldiffmerge,.

[AME00] M. Altheim and S. McCarron (EDS). XHTML 1.0: The Extensible HyperText Markup Language. W3C Recommendation, 2000. Available online at http://www.w3.org/TR/xhtml1.

[Bas97] P. G. Bassett. *Framing software reuse: lessons from the real world.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[Bat] D. Batory. AHEAD Tool Suite. web page. Available online at http://www.cs.utexas.edu/users/schwartz/ATS.html.

[Bat04] D. Batory. Feature-Oriented Programming and the AHEAD Tool Suite. In *Proceedings of the 26th International Conference on Software Engineering*, pages 702–703, May 2004.

[BCF03] V Benzaken, G. Castagna, and A. Frisch. CDuce: an XML-centric general-purpose language. In *In Proceedings of the 8th ACM SIGPLAN international conference on Functional programming, ICFP '03*, pages 51–63. ACM, 2003.

[BCP01] A. Bonifati, S. Ceri, and S. Paraboschi. Active Rules for XML: A New Paradigm for E-services. *The VLDB Journal*, 10(1):39–47, August 2001.

[Bea06a] J. Barnett et al. State Chart XML (SCXML): State Machine Notation for Control Abstraction. W3C Working Draft, 2006. Available online at http://www.w3.org/TR/scxml/.

[Bea06b] S. Boag et al. XQuery 1.0: An XML Query Language. W3C Candidate Recommendation, June 2006. Available online at http://www.w3.org/TR/xquery/.

[Bec02] M. Becker. XML-Enhanced Product Family Engineering. In *Proceedings of the 6th World Conference on Integrated Design and Process Technology (IDPT'02)*, 2002.

[BHFA04] B. Bouchou and M. Halfeld Ferrari Alves. Updates and Incremental Validation of XML Documents. In Georg Lausen and Dan Suciu, editors, *Database Programming Languages*, volume 2921 of *Lecture Notes in Computer Science*, pages 216–232. Springer Berlin Heidelberg, 2004.

[BKK03] M. Bernauer, G. Kappel, and G. Kramler. Approaches to Implementing Active Semantics with XML Schema. *23rd International Workshop on Database and Expert Systems Applications*, 0:559, 2003.

[BML$^+$04] D. Barbosa, A. O. Mendelzon, L. Libkin, L. Mignet, and M. Arenas. Efficient incremental validation of XML documents. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*. IEEE, 2004.

[BMS05] G. Bierman, E. Meijer, and W. Schulte. The essence of data access in Cw. In *Proceedings of the 19th European Conference on Object-Oriented Programming (ECOOP'05)*, volume 3586 of *Lecture Notes in Computer Science*. Springer-Verlag, July 2005.

[Bou03] R. Bourret. XML and Databases, 2003. Available online at http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[BPSM98] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, February 1998. Available online at http://www.w3.org/XML/.

[BSR04] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, June 2004.

[CB98] T. Connolly and C. Begg. *Database Systems*. Addison Wesley, 1998.

[CD99] J. Clark and S. DeRose. XML Path Language (Xpath) Version 1.0. W3C Recommendation, 1999. Available online at http://www.w3.org/TR/xpath.

[CFB00] S. Ceri, Piero Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33(1-6):137–157, 2000.

[Cla99] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, 1999. Available online at http://www.w3.org/TR/xslt.

[CM89] I. Amy Chen and D. McLeod. Derived Data Update in Semantic Databases. In *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB'89)*, pages 225–235, Amsterdam, The Netherlands, August 1989.

[CN01] P. Clements and L.M. Northrop. *Software Product Lines - Practices and Patterns*. Addison-Wesley, 2001.

[Con99] J. Conallen. Modeling Web Application Architectures with UML. *Communications of the ACM*, 42(10):63–70, 1999.

[Con00] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, 2000.

[Cor02] Microsoft Corporation. XML Diff and Patch, 2002. Available online at http://www.gotdotnet.com/xmltools/xmldiff/.

[DA02] O. Díaz and F. I. Anfurrutia. XDerive: a namespace for defining derived elements in XML Schema. In *Proceedings of the Workshop on Métodos y Herramientas para el comercio electrónico (ZOCO'02 co-located with JISBD'02)*, El Escorial, Madrid, Spain, 2002.

[DA05] O. Díaz and F. I. Anfurrutia. Improving self-interpretation of XML-based business documents by introducing derived elements. *Electronic Commerce Research and Applications (ECRA)*, 4:264–282, 2005.

[DAK09] O. Díaz, F. I. Anfurrutia, and J. Kortabitarte. Using DITA for documenting Software Product Lines. In *Proceedings of the 9th ACM symposium on Document engineering (DocEng'09), 231-240*, 2009.

[DCB09] B. Delaware, W. R. Cook, and D. Batory. Fitting the Pieces Together: A Machine-checked Model of Safe Composition. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 243–252, New York, NY, USA, 2009. ACM.

[DII00] O. Díaz, J. Iturrioz, and F. Ibáñez. Integración, navegación, presentación: experiencias utilizando XML. *Novatica*, (146):12–19, 2000.

[DII01a] O. Díaz, F. Ibáñez, and J. Iturrioz. A client intensive, a model-based approach to web application development: The AtariX system. In *Proceedings of the Workshop on Ingeniería del Software orientada a la Web (co-located with JISBD'01)*, Almagro, Ciudad Real, Spain, 2001.

[DII01b] O. Díaz, F. Ibáñez, and J. Iturrioz. A model-based approach to web-application development. In Robert Meersman, Karl Aberer, and Tharam Dillon, editors, *Semantic Issues in e-commerce systems IFIP TC2 / WG2.6 Ninth Working Conference on Database Semantics*, volume 111 of *IFIP - The International Federation for*

Information Processing, pages 295–309, Hong Kong, April 25–28 2001. Kluwer Academic Publishers.

[DIRA02] O. Díaz, A. Irastorza, J. J. Rodríguez, and F. I. Anfurrutia. An Overview on XML initiatives to Bring Modularization to Web Application Development. In *Proceedings of the WWW/Internet 2002 (IADIS International Conference)*, pages 435–443, 2002.

[DMOT01] S. DeRose, E. Maler, D. Orchard, and B. Trafford. XML Linking Language (XLinking) Version 1.0, 2001. Available online at http://www.w3.org/TR/xlink/.

[DRPI02] O. Díaz, J. J. Rodríguez, I. Paz, and F. Ibáñez. Wrapping HTML pages as Interactive Web Services. In *Proceedings of the Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'02)*, pages 301–309, 2002.

[DTA05] O. Díaz, S. Trujillo, and F. I. Anfurrutia. Supporting production strategies as refinements of the production process. In *Proceedings of the 9th International Software Product Line Conference (SPLC'05)*, volume 3714 of *Lecture Notes in Computer Science*, pages 210–221, Rennes, France, September 2005. Springer-Verlag.

[DVDHT05] E. M. Dashofy, A. E. Van Der Hoek, and R. N. Taylor. A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology*, 14(2):199–245, April 2005.

[Dör09] J. Dörre. Feature-Oriented Composition of XML Artifacts. Master's thesis, University of Passau, 2009.

[ea97] G. Kiczales et al. Aspect-Oriented Programming. In *Proceedings of ECOOP*, 1997.

[edi04] EDIFACT: Home Page, 2004. Available online at http://www.unece.org/cefact/edifact/welcome.html.

[Eur04] European Comission. IDA e-procurement protocol XML schemas initiative, 2004. Available online at http://europa.eu.int/ISPO/ida/export/files/en/1996.pdf.

[FP00] P. Fraternali and P. Paolini. Model-driven Development of Web Applications: The AutoWeb System. *ACM Trans. Inf. Syst.*, 18(4):323–382, October 2000.

[GC03] J. Gómez and C. Cachero. OO-H Method: Extending UML to Model Web Interfaces. In Patrick van Bommel, editor, *Information Modeling for Internet Applications*, chapter OO-H Method: Extending UML to Model Web Interfaces, pages 144–173. IGI Publishing, Hershey, PA, USA, 2003.

[Gea00] D. Geary. JSP Templates, 2000. Available online at http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-jspweb.html.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[GLC99] B. N. Grosof, Y. Labrou, and H. Y. Chan. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In *Proceedings of the 1st ACM Conference on Electronic Commerce (EC'99)*, pages 68–77, 1999.

[GM01] A. Ginige and S. Murugesan. Web Engineering: An Introduction. *IEEE MultiMedia*, pages 15–18, January - March 2001.

[GP98] C.F. Goldfarb and P. Prescod. *The XML Handbook*. Prentice Hall, Inc., 1998.

[GP03] V. Gapeyev and B. C. Pierce. Regular Object Types. In *Proceedings of the 17th European Conference on Object-Oriented Programming (ECOOP'03)*, pages 151–175, 2003.

[GPS93] F. Garozotto, P. Paolini, and D. Schwabe. HDM - A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems*, 11(1):1–26, 1993.

[GSG00] M. Gaedke, C. Segor, and H. W. Gallerse. WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. In *Proceedings of the ACM Symposium on Applied Computing (SAC'00)*, Villa Olmo, Como, Italy, March 19-21 2000.

[GWG97] H.-W. Gellersen, R. Wicke, and M. Gaedke. WebComposition: an object-oriented support system for the Web engineering life-cycle. *Computer Networks and ISDN Systems*, 29(8-13):1429–1437, September 1997.

[HBD$^+$04] D. Hirtle, H. Boley, C. Damassio, B. Grosof, S. Tabet, and G. Wagner. Specification of RuleML 0.86, 2004. Available online at http://www.ruleml.org/0.86/.

[HBH03] W. Hümmer, A. Bauer, and G. Harde. XCube: XML for data warehouses. In *Proceedings of the 6th International Workshop on Data Warehousing and OLAP(DOLAP'03)*, pages 33–40, New Orleans, Louisiana, USA, November 2003. ACM.

[Hea99] R. Hull et al. Declarative Workflows that Support Easy Modification and Dynamic Browsing. In *Proceedings of the ACM International Joint Conference on Work Activities Coordination and Collaboration (WACC'99)*, pages 69–78, 1999.

[HP03] H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Trans. Inter. Tech.*, 3(2):117–148, 2003.

[HRS$^+$05] M. Harren, M. Raghavachari, O. Shmueli, M. G. Burke, R. Bordawekar, I. Pechtchanski, and V. Sarkar. XJ: facilitating XML processing in Java. In *Proceedings of the 14th international conference on World Wide Web (WWW'05)*, pages 278–287, New York, NY, USA, 2005. ACM Press.

[HVP05] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(1):46–90, January 2005.

[IDR02a] F. Ibáñez, O. Díaz, and J. J. Rodríguez. Coarse-grained delivery units: from HTML pages to XML Leaflets. In *Proceedings of the Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 311–318, Madrid, 2002.

[IDR02b] F. Ibáñez, O. Díaz, and J. J. Rodríguez. Extending XML Schema with Derived Elements. In *Proceedings of the IFIP WG8.1 Working Conference on Engineering Information Systems in the Internet Context*, volume 231 of *IFIP Conference Proceedings*, pages 53–67. Kluwer Academic Publishers, 2002.

[IS92] Y. E. Ioannidis and T. K. Sellis. Supporting Incosistent Rules in Database Systems. *Journal of Intelligent Information Systems*, (1):243–270, 1992.

[ISB95] T. Isakowitz, E. A. Stohr, and P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8):34–43, 1995.

[JA01] R. Jelliffe and Academia Sinica Computing Centre. The Schematron: An XML Structure Validation Language using Patterns in Trees. Web page, 2001. Available online at http://www.ascc.net/xml/resource/schematron/schematron.html.

[JCP03] JCP. JSR 168 Portlet Specification Version 1.0, September 2003. Available online at http://www.jcp.org/en/jsr/detail?id=168.

[JP14] Paul Johannesson and Erik Perjons. *An introduction to design science*. Springer, 2014.

[KAT$^+$09] C. Kästner, S. Apel, S. Trujillo, M. Kuhlemann, and D. Batory. Guaranteeing Syntactic Correctness for All Product Line Variants:

A Language-Independent Approach. In Manuel Oriol and Bertrand Meyer, editors, *Objects, Components, Models and Patterns*, volume 33 of *Lecture Notes in Business Information Processing*, pages 175–194. Springer Berlin Heidelberg, 2009.

[Kea90] K. Kang et al. Feature Oriented Domain Analysis (FODA) Feasability Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990.

[KKW05] K. Kelly, J. J. Kratky, and K. Wells. Model Driven Compound Document Development. In *Proceedings of the XTech 2005: XML, the Web and beyond*, 2005.

[KL02] M. Kempa and V. Linnemann. On XML Objects. In *Informal Proceedings of the Workshop on Programming Language Technologies for XML (PLAN-X'02)*, pages 44–54, Pittsburgh, USA, 2002.

[KM06] C. Kirkegaard and A. Möller. Type checking with XML Schema in XACT. In BRICS, editor, *Informal Proceedings of the PLAN-X 2006*, number NS-05-6, Charleston, South Carolina, January 2006.

[KNC01] R. Klapsing, G. Neumann, and W. Conen. Semantics in Web Engineering: Applying the Resource Description Framework. *IEEE Multimedia*, 8(2):62–68, April-June 2001.

[Koc01] N. Koch. *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process*. PhD thesis, Ludwig-Maximilians-Universität München, 2001.

[KP88] G.E. Krasner and S.T. Pope. A Cookbook for Using the Model-View-Controller Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, pages 20–30, August-September 1988.

[KSR02] B. Kane, H. Su, and E. A. Rundensteiner. Consistently Updating XML Documents using Incremental Constraint Check Queries. McLean, Virginia, USA, November 2002. ACM.

[LF01] R. La Fontaine. A Delta Format for XML: Identifying changes in XML and representing the changes in XML. In *XML Europe*, 2001.

[LM00] A. Laux and L. Martin. XML Update Language. XML:DB Working Draft, September 2000. Available online at http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html.

[MCV04] P. Marinelli, C.S. Coen, and F. Vitali. SchemaPath, a Minimal Extension to XML Schema for Conditional Constraints. In *Proceedings of the 13th International World Wide Web Conference*, pages 164–174, New York, USA, 2004.

[Mey97] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall PTR, second edition, March 1997.

[Mey14] B. Meyer. *Agile! - The Good, the Hype and the Ugly*. Springer, 2014.

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-Specific Languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.

[NJB03] C-K. Nam, G-S. Jang, and J-H. J. Bae. An XML-based active document for intelligent web applications. *Expert Systems with Applications*, 25(2):165 – 176, 2003.

[NKAM09] M. Niederhausen, S. Karol, U. Aßmann, and K. Meißner. HyperAdapt: Enabling Aspects for XML. In Martin Gaedke, Michael Grossniklaus, and Oscar Díaz, editors, *Web Engineering*, volume 5648 of *Lecture Notes in Computer Science*, pages 461–464. Springer Berlin Heidelberg, 2009.

[OAS04] OASIS. SGML/XML: Using Elements and Attributes. OASIS Cover Pages, 2004. Available online at http://xml.coverpages.org/elementsAndAttrs.html.

[Obj] Object Management Group. XML Meta Data Interchange (XMI) 2.0. Available online at http://www.omg.org.

[Obj05] Object Management Group. Reusable Asset Specification, November 2005. Available online at http://www.omg.org/docs/formal/05-11-02.pdf.

[Ogb04] U. Ogbuji. Principles of XML design: When to use elements versus attributes. Exploring the Oldest Question in XML Design. IBM developerWorks, 2004. Available online at http://www-128.ibm.com/developerworks/xml/library/x-eleatt.html.

[OOP] OOPs Consultancy. XmlTask. Available online at http://www.oopsconsultancy.com/software/xmltask/index.html.

[Ora02] Oracle Corporation. The Oracle´s XML Parser for Java, 2002. Available online at http://otn.oracle.com/tech/xml/xdk_java/content.html.

[PBvdL06] K. Pohl, G. Bockle, and F. van der Linden. *Software Product Line Engineering - Foundations, Principles and Techniques*. Springer, 2006.

[PD99] N. W. Paton and O. Díaz. Active Database Systems. *ACM Comput. Surv.*, 31(1):63–103, March 1999.

[PM00] F. Paterno and C. Mancini. Model-Based Design of Interactive Applications. *ACM Intelligence*, pages 27–37, Winter 2000.

[Pre97] C. Prehofer. Feature-Oriented Programming: A Fresh Look At Objects. In *Proc. European Conf. Object-Oriented Programming*, pages 419–443. Springer, 1997.

[PV03] Y. Papakonstantinou and V. Vianu. Incremental Validation of XML Documents. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *Database Theory — ICDT 2003*, volume 2572 of *Lecture Notes in Computer Science*, pages 47–63. Springer Berlin Heidelberg, 2003.

[RD01] J.J. Rodríguez and O. Díaz. Seamless Integration of Inquiry and Transaction Tasks in Web Applications. In Robert Meersman, Karl Aberer, and Tharam Dillon, editors, *Semantic Issues in e-commerce Systems*, IFIP Conference Proceedings. Kluwer Academic Publishers, 2001.

[RDA02] J. J. Rodriguez, O. Diaz, and F. I. Anfurrutia. Moving Web Services Dependencies at the front-end. In *Proceedings of the IFIP WG8.1 Working Conference on Engineering Information Systems in the Internet Context*, volume 231 of *IFIP Conference Proceedings*, pages 221–237. Kluwer Academic Publishers, 2002.

[RJ05] D. C. Rajapakse and S. Jarzabek. An investigation of cloning in web applications. In *Proceedings of the 5th International conference on Web Engineering (ICWE'05)*, Lecture Notes in Computer Science, pages 252 – 262, Sydney, Australia, 2005. Springer.

[SB02] M. Schrefl and M. Bernauer. Active XML Schemas. In Hiroshi Arisawa, Yahiko Kambayashi, Vijay Kumar, HeinrichC. Mayr, and Ingrid Hunt, editors, *Conceptual Modeling for New Information Systems Technologies*, volume 2465 of *Lecture Notes in Computer Science*, pages 363–376. Springer Berlin Heidelberg, 2002.

[SDRI02a] S. Steinau, O. Díaz, J. J. Rodríguez, and F. Ibáñez. A tool for assesing the consistency of Websites. In *Proceedings of the 4th International Conference on Enterprise Information Systems (ICEIS'02)*, volume 2, pages 691–698, 2002.

[SDRI02b] S. Steinau, O. Díaz, J. J. Rodríguez, and F. Ibáñez. *Enterprise Information Systems IV*, chapter A tool for assesing the consistency of Websites, pages 227–234. Kluwer Academic Publishers, 2002.

[Ses11] Maider Azanza Sesé. *Model Driven Product Line Engineering: Core Asset and Process Implications*. PhD thesis, University of the Basque Country (UPV/EHU), 2011.

[SK05a] S. Speicher and K. E Kelly. Compound XML document profiles for rich content, Part 1: Exploring extensibility alternatives using XML Schema, 2005. Available online at http://www-128.ibm.com/developerworks/xml/library/x-cxdp1/.

[SK05b] S. Speicher and K. E Kelly. Compound XML document profiles for rich content, Part 2: A pattern for developing compound XML document schemas, 2005. Available online at http://www-128.ibm.com/developerworks/xml/library/x-cxdp2/.

[SR98] D. Schwabe and G. Rossi. An object oriented approach to Web-based applications design. *Theory and Practice of Object Systems*, 4(4):207–225, 1998. John Wiley & Sons, Inc.

[SSJ02] I. Singh, B. Stearns, and M. Johnson. *Designing Enterprise Applications with the J2EE Platform.* Addison-Wesley, 2002.

[Suc02] D. Suciu. The XML typechecking problem. *SIGMOD Rec.*, 31(1):89–96, March 2002.

[Sun] Sun Microsystems Inc. Java API for XML Processing (JAXP). Available online at http://java.sun.com/xml/jaxp/.

[SWGZ00] M. W. Schranz, J. Weidl, K. M. Goschka, and S. Zechgmeister. Engineering complex World Wide Web services with JESSICA and UML. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS'00)*, Maui, HI, USA, 2000.

[SYK$^+$10] Boshi Sun, Xiaojie Yuan, Hong Kang, Xiaocheng Huang, and Ying Guan. Incremental Validation of XML Document Based on Simplified XML Element Sequence Pattern. In *Web Information Systems and Applications Conference (WISA), 2010 7th*, pages 110–114, Aug 2010.

[SZJ02] S. M. Swe, H. Zhang, and S. Jarzabek. XVCL: a tutorial. In *Proceedings of the 14th International Conference on Software En-*

gineering and Knowledge Engineering (SEKE'02), pages 341–349, New York, NY, USA, 2002. ACM Press.

[TBD06] S. Trujillo, D. Batory, and O. Díaz. Feature Refactoring a Multi-Representation Program into a Product Line. In *Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE'06)*, Lecture Notes in Computer Science, 2006.

[TBD07] S. Trujillo, D. Batory, and O. Díaz. Feature Oriented Model Driven Development: A Case Study for Portlets. In *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, Minnesota, USA, May 20-26*, 2007.

[TBKC07] S. Thaker, D. Batory, D. Kitchin, and W. Cook. Safe Composition of Product Lines. In *Proceedings of the 6th International Conference on Generative Programming and Component Engineering*, GPCE '07, pages 95–104, New York, NY, USA, 2007. ACM.

[TBMM01] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. W3C Recommendation, 2001. Available online at http://www.w3.org/TR/xmlschema-1/.

[Thea] The Apache Software Foundation. Apache Ant. Available online at http://ant.apache.org.

[Theb] The Apache Software Foundation. Apache Struts. Available online at http://struts.apache.org/.

[The01] The Apache Software Foundation. Xerces2 Java Parser, 2001. Available online at http://xml.apache.org/xerces2-j/index.html.

[TL98] O.M.F. De Troyer and C.J. Leune. WSDM: a user centered design method for Web sites. *Computer Networks and {ISDN} Systems*, 30(1):85 – 94, 1998. Proceedings of the Seventh International World Wide Web Conference.

[ubl04] Online community for the Universal Business Language (UBL). OASIS Standard, 2004. Available online at http://ubl.xml.org.

[Urp06] J. Urpalainen. An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors. IETF SIMPLE WG,Internet-Draft 'draft-urpalainen-simple-xml-patch-ops-02', 2006.

[VdV00] E. Van der Vlist. Style-free XSLT Style Sheets, 2000. Available online at http://www.xml.com/pub/a/2000/07/26/xslt/xsltstyle.html.

[W3C02] W3C. XML Inclusions (XInclude) Version 1.0. W3C Recommendation, 2002. Available online at http://www.w3.org/TR/xinclude/.

[W3C04] W3C. Compound Document Formats. W3C Recommendation, 2004. Available online at http://www.w3.org/2004/CDF/.

[Wea98] L. Wood and et al. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, 1998. Available online at http://www.w3.org/TR/REC-DOM-Level-1.

[Wir71] N. Wirth. Program Development by Stepwise Refinements. *Communications of the ACM*, 14(4):221–227, April 1971.

[WP11] N. Wahid and E. Pardede. XML semantic constraint validation for XML updates: a survey. In *Proceedings of International Conference on Semantic Technology and Information Retrieval (STAIR)*, pages 57–63. IEEE, 2011.

[x1204] ASC X12 The Accredited Standards Committee Home Page, 2004. Available online at http://www.x12.org.

[Zhu03] H. Zhuge. Active e-document framework ADF: model and tool. *Information & Management*, 41(1):87 – 97, 2003.

Acknowledgements

I would like to thank many people who has made possible, directly or indirectly, the materialization of this thesis.

First and foremost, I would like to thank my advisors Prof. Oscar Díaz for his kind support and much-needed guidance. It was him who sparked my interest in research and helped me appreciate the value of looking beyond the horizon of the immediately practical. I am quite fortunate to have had him as an advisor, and I am very grateful for all of his help and patience.

During this study, I enjoyed working together with the members of the ONEKIN research group, which Oscar leads. I am indebted to all of my workmates, specifically Jon Iturrioz, Juanjo Rodríguez, Iñaki Paz and Mikel Larrañaga taking part in the development of *XLeaflet* and Salvador Trujillo with respect to the *XAK* project, part of this work would not be possible without their invaluable discussions, efforts and advices. Thanks to Sergio F. Anzuola for both his kindness and cheering up the atmosphere of the isolated Arbide Towers (the working place). I am also grateful with the former and actual remaining members of ONEKIN: Luis M. Alonso, Maider Azanza, Iker Azpeitia, Oscar Barrera, Cristobal Carellano, Jose Ramón Díaz, Arantza Irastorza, Arturo Jaime, Jon Kortabitarte, Felipe Martin, Sandy Perez, Gorka Puente, Itziar Otaduy, Leticia Montalvillo, Iñigo Aldalur, and the newbie Jeremias Perez, who are inclined to help when you need it.

I would like to express my gratitude to the University of the Basque Country, to the Department of Computer Languages and Systems and specifically, to my colleagues at EUI in Vitoria-Gasteiz: Ainhoa Alvarez, Ismael Etxeberria, Borja Fernandez, Pablo Gonzalez, Mikel Larrañaga, Pablo Navarro, Mari Carmen Otero, Patxi Ramirez and Iñigo Quintana, for allowing my liberation of teaching work in order to finish the thesis writing.

Other old friends of lab 315 who have stayed in touch have often helped to provide many invaluable and enlightening conversations and essential diversions: Amaia, Ainhoa, Patricia, Yolanda, Jimena, David, Mili, Gus, Alberto and Licri. Although the thesis work requires a full time of dedication, from time to time there is a need to take the air for an inspiration and accept an invitation of closed friends in order to distract from the thesis matter. Thanks to all of them, my *cuadrilla*.

Thanks to whole my family for understanding me, giving their support and being always there when was necessary. But especially thanks to my parents and parents-in-law, who takes care of our little girls. Eskerrik asko bihotz-bihotzez.

Last but not least, I am especially grateful to my wife Miren for her constant support and encouragement, patience and *love*. She has endured many years of partial attention and cancelled holidays in order that I might reach this point. I am lucky to have such an understanding wife. Eta nola ez, nere bi printzesei, Izaro eta Loreari, zuen alaitasun eta bizi-pozarengatik.

Summary

The presence of XML is pervasive, yet its youth makes developers face a lot of challenges when using XML in cutting-edge applications. This thesis confronts XML in three different scenarios: document exchange, Software Product Lines (SPLs) and Domain-Specific Languages (DSLs). Digital document interchange is one of the prominent applications of XML. However, business documents frequently hold derived data, i.e. data which is calculated from other data. Here, we face the question of how can XML Schema be extended to account for derived data. This dissertation proposes the XDerive namespace that permits deriving functions to be integrated as part of XML Schema documents.

Next, SPLs offer an approach to develop a family of software products by reuse. What if these products are realised as XML documents? The thesis addresses how Feature-Oriented Programming (an approach to SPL development) can be extended to account for XML specifics (e.g. tag-based description, validation awareness).

Finally, XML in Web development. The increasing growth in size and complexity of web sites calls for a systematic way to web sites development. Leaflet websites are a kind of content-oriented websites. Domain Specific Languages (DSL) are usually geared towards a specific domain or application, offering only a restricted suite of notations and abstractions. Here, we address the domain of leaflet websites, i.e. websites meant for static-content navigation, and introduce an XML-based DSL: XLeaflet. A distinctive feature of XLeaflet is its architecture: thick-browser. This can account for an important reduction in the network traffic. These so-different fields (i.e. document exchange, SPLs and DSLs) act as “stress tests” to assess the ductility of XML concepts and technology to cope with so heterogeneous environments.